МОНИТОР

версия 2.91

ISSN 0869-3889

защита данных IBM PC

язык РЕФАЛ

фирма APPLE -стратегия развития

политика NANTUCKET

графика TURBO PASCAL

Все о TURBO C и C++!

"Библиотека журнала "Монитор"
представляет первую книгу

"Функции Turbo C, C++".

Более чем на 300 страницах тщательно подобрана информация о библиотеках C, Turbo C, C++.

Вы получите:

- детальное описание более 400 функций;
- примеры, поясняющие работу каждой функции;
- список совпадающих функций Turbo C, C++, Microsoft C;
- демонстрационные программы, поясняющие работу групп функций;
- указатели для быстрого поиска по имени функции или ее назначению;
- введение в объектно-ориентированные средства Turbo C++;
- современный дизайн и твердый переплет.

Эта книга может служить справочником и учебником. Мы позаботились о том, чтобы Вам было удобно и приятно ее читать.

Если вы программируете на Turbo C и C++, интересуетесь вопросами совместимости с Microsoft C, то эта книга для Вас.

Спешите! Тираж ограничен.

Ориентировочная цена книги 24 рубля
Оптовым покупателям предоставляются скидки.
Заказы направлять по адресу: 117419, Москва,
а/я 765, "Монитор"

Вышлите 3 экз. кни
„Функции Turbo C

ПОЧТОВАЯ КАРТОЧКА

Куда 117419, г. Москв
а/я 765

Кому „Монитор"

117419

Пишите индекс предприятия связи места назначен

МОНИТОР

Журнал для программистов - практиков

Редколлегия:

Д.Э.Бройтман
И.Г.Гун
(главный редактор)
И.Н.Исаев
С.В.Макельский
(главный художник)
А.В.Орехов
(издательский директор)
М.В.Остринский
Л.И.Устинова
А.А.Якубенко
(ответственный секретарь)

В оформлении использованы рисунки К.З. Гусалова

Подписано в печать с оригинал-макета 26.12.91.
Формат 60x84/8.
Печать офсетная. Печ.л.6.0.
Изд. N1971.
Цена договорная.

Издатель:
Научно-техническое общество

Софт·Москва

Для писем: 117419, г.Москва, а/я 765, тел/факс (095)126-31-56

© "Монитор" 2,1991

Технологии программирования	М. Черкашин Вкусные и невкусные рыбы в объектно-ориентированном программировании	2
Компьютер в разрезе	А. Тихонов Что-то с памятью моей стало (часть 2)	9
Не только PC	В. Крюков, С.Пучков Куда катится компьютерное яблоко	15
Языки программирования	И. Скрипачев Язык Рефал - первое знакомство	21
Панорама	История языков программирования	28
Лицей	И.Гун Мы дружим с графикой ! или введение в использование библиотеки Turbo Pascal	30
Компьютер и право	М.Корсаков Юридическая защита программ	39
	В.Водолазский Copyright в вашей программе	45
ДОС под микроскопом	В.Аношин Защита программ и данных на IBM PC	49
Панорама	Сленг программистов-хэкеров США	55
Экспертные системы	А. Свириденко Интерфейсы- путь к взаимопониманию	56
Компьютерный бизнес	А.Орехов Nantucket делает ставку на советский рынок	61
Панорама	Из книги рекордов Гиннеса	64

М. Черкашин

Вкусные и невкусные рыбы в объектно–ориентированном программировании

А что такое объектно-ориентированное программирование вообще? В самом примитивном своем варианте - всего несколько новых конструкций к традиционным языкам программирования. Однако существуют и менее традиционные возможности.

КАКИМ БЫВАЕТ НАСЛЕДОВАНИЕ

Известный персонаж Жюль Верна делил, как известно, рыб на два класса : съедобные и несъедобные. Мы же можем усовершенствовать его классификацию и съедобных рыб делить в свою очередь на вкусных и невкусных (рис.1а). И какую бы мы область не взяли, там всегда найдется какая-нибудь классификация. А в любой классификации более узкий класс изучаемых объектов всегда получается уточнением более широкого.

Трудно сказать, чего больше в этом обилии различных классификаций - действительно необходимости или пристрастия человека к раскладыванию по полочкам собственных знаний.

Кто-нибудь мог придумать другую классификацию рыб - например, на рыб первого сорта и второго (рис.1б). Так получаются вполне вкусные рыбы (рис.1в) - те, которые и вкусные и свежие.

Любая классификация предполагает, что в каждом частном случае о предмете можно сказать что-то такое, что неверно в более общем. Это относится как к рыбам, так и к программированию. (Так, только съедобную рыбу можно готовить, а затем есть; в программировании конкретизации соответствует наследование).

Попробуем, например, с помощью наследования организовать работу с простейшей базой данных.

Начнем с наброска (пример 1), а затем, используя наследование, уточним некоторые детали и получим две разновидности базы данных, которые можно использовать одновременно.

Нам известно, что база данных - это, вообще говоря, произвольная последовательность записей, каким-то образом организованная. В некоторых записях может содержаться полезная информация. Другие же, напротив, пока свободны (случай ПУСТО = true) и готовы для размещения там новых порций элементов данных.

Для каждой записи существует ключ. Внимание ! По ключу мы будем каждый раз искать запись в базе данных, и поэтому выбор ключа очень важен. Ключ должен быть уникален и иметь смысл для человека, который использует систему. Например, название цеха является хорошим примером ключа. Ключами не следует делать фамилию начальника

цеха (а вдруг двумя цехами управляют однофамильцы), адрес цеха (очень неудобно вспоминать адрес цеха всякий раз перед обращением к программе). Существует процедура размещения элемента X в базе данных. Таким образом, с потомком мы сможем сделать все то же, что и с предком, и даже более.

Здесь используются два различных способа организации доступа. Индексирование с использованием хэ-

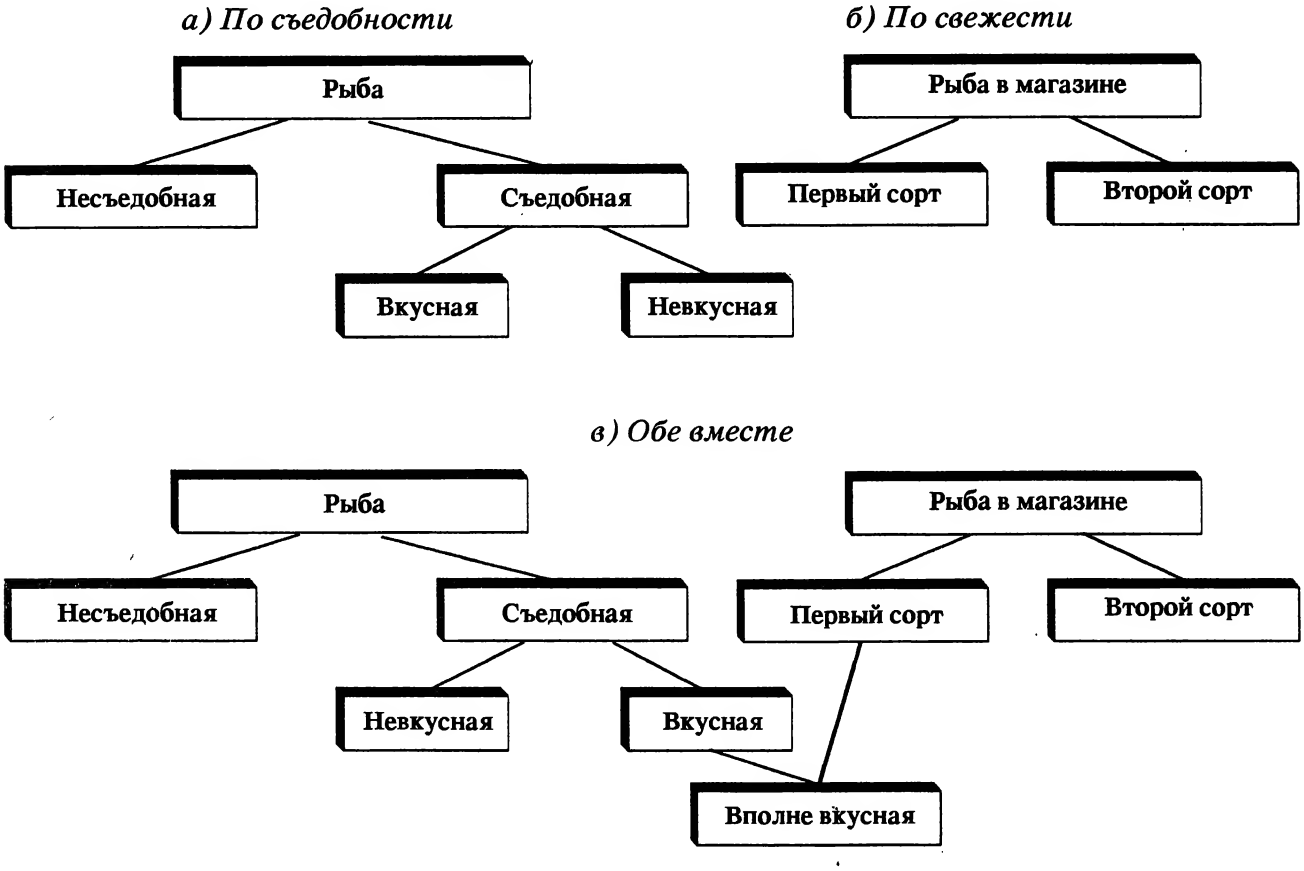


Рис. 1. Три различных классификаций рыб

Она включает в себя поиск по ключу записи, куда этот элемент должен быть помещен, и размещение элемента в определенной записи. Поиск места, куда записывается новый элемент, может быть различным при различной организации базы данных. В качестве ключей могут служить элементы различных типов, однако здесь (как часто и бывает) используются строки. Элементами базы данных служат элементы некоторого уже известного типа ТИПЗАПИСИ.

Но DataBase - это только макет базы данных. Про нее еще почти ничего не известно. Прежде чем реально использовать такую базу данных, следует уточнить, как мы будем к ней обращаться. *Такое уточнение в объектно-ориентированном программировании называется наследованием.* Название само подсказывает способ, которым мы будем уточнять то, что уже известно нам (и машине). Потомок получает в свое распоряжение все данные и процедуры, имеющиеся у предка.

ширования основано на следующей идее. Для каждого ключа заранее определяем, в какой записи базы данных следует храниться записям с этим ключом. Однако записей существенно меньше, чем возможных ключей. Поэтому может случиться так, что ячейка, в которую мы хотели записать новый элемент, уже занята. Тогда мы стараемся поместить его возможно более близко к этой ячейке, т.е. справа от нее. При поиске элемента в базе данных мы сначала смотрим его место, потом просматриваем все записи справа от него подряд, пока не наткнемся на этот элемент либо на пустую запись. Если мы встретили пустую запись, то элемента в базе данных нет - если бы он был, то должен быть записан здесь.

Поиск "своего" места для каждого ключа производится с помощью хэш -функции Hash. Она должна просто вычисляться и равномерно раскидывать элементы по всей базе данных. Так, не следует определять хэш-функцию, исходя из первого символа ключа (подобно

```
const
  Объем = 1024;

type
  Запись = object
  Ключ : string;
  Пусто : boolean;
  ЭлементДанных : ТипЗаписи;
  function ПустоЛи : boolean;
end;

DataBase = object
  Память : array [1..Объем] of Запись;
  procedure ПоложитьЭлемент (Ключ : string; X : ТипЗаписи);
  function КудаКласть (Ключ: string): integer; virtual;
  function НайтиПоКлючу (Ключ : string) : integer; virtual;
end;

procedure DataBase.ПоложитьЭлемент;
var
  Номер : integer;
begin
  Номер := КудаКласть (Ключ);
  Память [Номер] := X;
  ПоложитьЭлемент := Номер;
end;
```

Пример 1 .Простейшая база данных

тому, как это сделано в записной книжке) - скорее всего, ключи на букву П встречаются куда чаще, чем ключи на букву Ю. (В самом деле, много ли у вас знакомых с фамилией на букву Ю). С помощью наследования можно надстроить скелет базы данных DataBase до такой базы данных (пример 2).

Существует и другой метод, основанный на том, что ключ каждой записи из базы заносится в отдельный список, и в этом списке указывается, где искать элемент с этим ключом. Ключи в этом списке упорядочены "по алфавиту", поэтому довольно легко найти ключ, а затем, пользуясь списком, и сам элемент.

Мы, конечно, могли бы поступить с искомыми элементами так же, как и в предыдущем методе доступа. Но дело в том, что работа с упорядоченными массивами встречается достаточно часто, и может быть даже в стандартной библиотеке объектно-ориентированного языка.

Допустим, в этой библиотеке (или в другом месте нашей программы) описан класс OrderedSet, осуществляющий работу с упорядоченными множествами строк (как раз такими, какие служат у нас ключами). И пусть в этом классе есть следующие процедуры:

1. Insert - для вставки в упорядоченный список строки и соответствующего ему числа.
2. Find - для поиска числа по строке (ключу).
3. InList - для проверки принадлежности данного ключа списку.
4. Size - для определения размера (т.е. числа элементов) списка.

Пример 2.База данных ХэшБаза

```
type
  ХэшБаза = object
  function Hash (Ключ : integer) : integer;
  function КудаКласть; virtual;
  function НайтиПоКлючу; virtual;
end;

(* В качестве хэш-функции используется сумма всех кодов
в строке. На практике это достаточно надежно. *)
function ХэшБаза.Hash;
var i, Сумма : integer;
begin
  Сумма := 0;
  for i:=1 to length (Ключ) do
  Сумма := Сумма + Ключ [i];
  Hash := Сумма mod Объем + 1;
end;

(* База данных "завернута в кольцо" так, чтобы у каждого
элемента были следующие *)
function След (Место : integer);
begin
  След := Место mod Объем +1;
end;

function ХэшБаза.КудаКласть;
var
  Место : integer;
begin
  Место := Hash(Ключ);
  while not ПустоЛи(Место) do
  Место := След(Место);
  КудаКласть := Место;
end;


function ХэшБаза.НайтиПоКлючу;
var
  Место : integer;
begin
  Место := Hash (Ключ);
  while not ПустоЛи(Место) and (Память[Место]Ключ) do
    Место := След (Место);
  if ПустоЛи (Место)
    then НайтиПоКлючу := 0 (* Элемент не найден *)
    else НайтиПоКлючу := Место;
end;
```

```
type

  УпорядБаза = object (OrderedSet, DataBase)
    function КудаКласть; virtual;
    function НайтивПоКлючу; virtual;
  end;

(* Число элементов в базе равно размеру списка
ключей. Поэтому элемент базы с номером Size+1
будет свободен *)

function УпорядБаза.КудаКласть;
var
  Место : integer;
begin
  КудаКласть := Size+1;
  Insert (Ключ, Size+1)
end;

function УпорядБаза.НайтиПоКлючу;
var
  Место : integer;
begin
  if InList (Ключ) then begin
    НайтиПоКлючу := Find (ключ);
  end
  else НайтивУпорядбазе := 0;
```

Пример 3.База данных УпорядБаза

И если мы объединим в одном объекте процедуры OrderedSet и базу данных DataBase, то мы получим то, что нам нужно (см. базу данных УпорядБаза в примере 3).

Класс УпорядБаза является наследником сразу двух классов - OrderedSet и DataBase. Данный механизм называется множественным наследованием.

Теперь мы можем создать классификацию типов баз данных, имеющихся в нашей программе. Эта классификация построена на "родственных отношениях" между ними, поэтому ее наглядное изображение часто называют генеалогическим деревом (рис.2).

Обратим внимание, как на этой диаграмме выделяются случаи множественного наследования - от класса DataBase идут две стрелки.

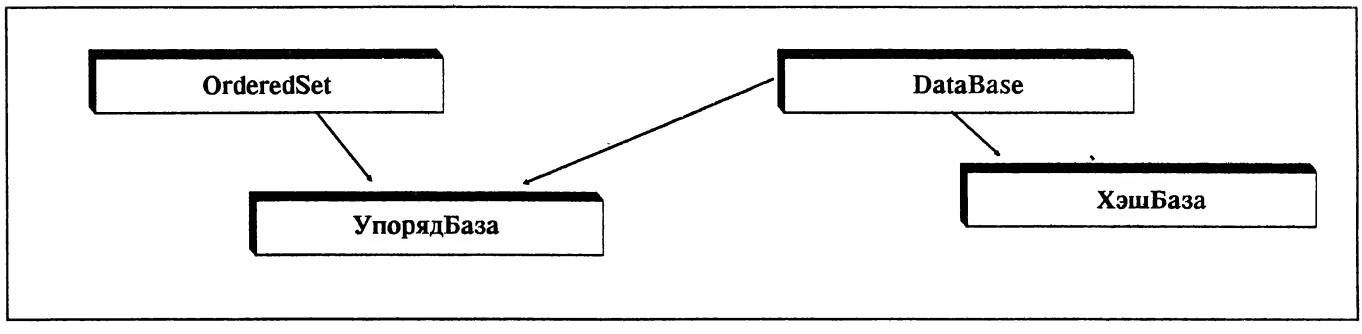


Рис. 2. Генеалогическое дерево баз данных

КОНСТРУКЦИЯ GENERIC

Рассмотрим один простой пример. В программировании часто используется структура данных, называемая стеком, который можно сравнить со стопкой бумаги, где листы кладутся только на ее вершину и оттуда же и убираются.

Легко написать комплект процедур для работы со стеком элементов любого заранее известного типа. Однако хотелось бы, чтобы одни и те же процедуры были применимы к стекам с различными типами элементов. Тогда мы получим пример абстракции "стека вообще", безотносительно к типу его элементов. К сожалению, в традиционных языках этого достичь невозможно. Для каждого типа стека весь комплект процедур приходится дублировать.

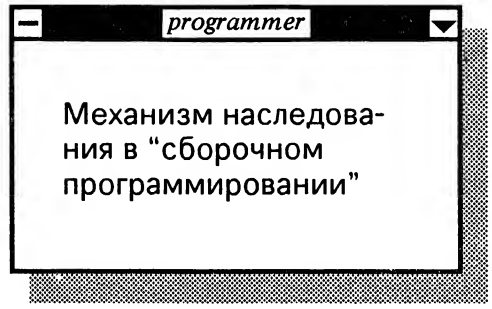
В некоторых языках (Ада, Эйфель) существует конструкция GENERIC, позволяющая справиться с этой проблемой. Пусть есть некоторый тип, а какой именно - нам пока неизвестно. Вместо него, подобно параметру в процедуре, будем использовать переменную X. Так, мы можем описать массив элементов типа X, отложив выяснение того, что же это за тип X, до лучших времен. Можем даже производить с этим неизвестным типом определенные манипуляции: присваивать значения этого типа, выдавать их как результат функции. В примере со стеком мы делаем все необходимые действия, не имея никакой информации о типе его элементов. В самом деле, что необходимо для организации стека? Переменная ВЕРШИНА, содержащая число элементов, хранящихся в данный момент в стеке, и массив ПАМЯТЬ для хранения этих элементов. В этом массиве элементы стека хранятся, начиная с ячейки ПАМЯТЬ, по ячейку ПАМЯТЬ [ВЕРШИНА]; последний элемент хранится в ячейке ПАМЯТЬ [ВЕРШИ-

```
const
  РАЗМЕР = 10000;

type

  Stack generic (ТипЭлемента) = object
    Память : array [1..Размер] of X;
    Вершина : integer;
    procedure Init;
    procedure Push (Элем : X);
    function Pop : Элем;
  end;

(* Инициализация стека *)
procedure Stack.Init;
begin
  Вершина := 0;
end;

(* Поместить элемент Элем на вершину стека *)
procedure Stack.Push;
begin
  Вершина := Вершина+1;
  Память [Вершина] := Элем;
end;

(* Взять элемент с вершины стека *)
function Stack.Pop;
begin
  Pop := Память [Вершина];
  Вершина := Вершина-1;
end;
```

Пример 4. Пример со стеком

НА]. Для работы со стеком необходимы три процедуры: подготовка стека к работе (инициализация), засылка элементов в стек и извлечение их оттуда. По традиции последние две процедуры называются английскими словами Push (затолкнуть) и Pop (вытолкнуть). Реализация стека показана в примере 4. Обратим внимание на конструкцию GENERIC (ТИПЭЛЕМЕНТА). Она сообщает компилятору , что переменная ТИПЭЛЕМЕНТА используется как параметр вместо неизвестного типа.

СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ GENERIC И НАСЛЕДОВАНИЯ



В объектно-ориентированном программировании очень важно понятие полиморфизма. Полиморфизм в программировании - это использование одного имени функции или обозначения операции для различных действий. Какое из этих действий будет выполняться в каждом конкретном случае, определяется из контекста, например, по типу операндов. Так, умножение целых и вещественных чисел - две различные операции; однако они обозначаются одним и тем же значком *. В объектно-ориентированном программировании одно имя может использоваться функциями из разных классов, если один из этих классов является потомком другого. Это и есть полиморфизм.

Вспомним конструкцию GENERIC (X). Предположим, что мы пытаемся ее использовать в языке, в котором полиморфизм отсутствует полностью. Тогда любая операция применима к данным только какого-то одного типа. При любой попытке хоть что-нибудь сделать с объектом типа X, мы должны применить к нему некую операцию; следовательно, тип, с которым она работает, и есть тип X. Зачем же нам тогда эта конструкция? Положение спасает то, что к любому типу применима операция присваивания. Таким образом, в нашем языке имеется встроенный, хотя и примитивный полиморфизм; такой полиморфизм необходим для использования конструкции GENERIC. Мы можем существенно обогатить возможности этой конструкции, если будем сами определять полиморфизм так, как нам нужно. Это можно делать с помощью конструкции наследования.

Для иллюстрации этого несколько изменим наш последний пример. Ранее мы предполагали, что тип X нам заранее известен, и ограничивались, таким образом, базами данных с элементами только этого типа X в качестве записей. Но это же искусственное и совершенно неоправданное ограничение. Нам достаточно написать несколько строк (пример 5), и база данных может содержать элементы любого типа - такого, какого мы захотим.

Совместное использование параметризованных типов (GENERIC) и наследования можно показать на плоскости (рис. 3). Пусть на этой плоскости присутствуют две координатные оси - ось GENERIC и ось наследования. Используя эти механизмы по отдельности, мы не сможем выйти за пределы этих двух осей. Совместному же использованию на этой диаграмме соответствует вся координатная плоскость.

ЧТО ДЕНЬ ГРЯДУЩИЙ НАМ ГОТОВИТ

Появление объектно-ориентированного программирования - это самый настоящий переворот в программировании. Но такие перевороты в истории программирования уже бывали. Более того, идеи, лежащие в его основе, появились довольно давно. Начнем со структурного программирования. Как оно появилось - разговор

```
type
  Запись GENERIC(ТипЗаписи)
    Ключ : string;
    Пусто : boolean;
    Элемент данных : ТипЗаписи;
    function ПустоЛи : boolean;
  end;
```

Пример 5. Совместное использование параметризованных типов и наследования

особый. Рассмотрим банальнейший случай - программу сортировки (пример 6).

Кажется, что все три процедуры в вышеописанной программе совершенно самостоятельны и не зависят друг от друга. Однако это не так. Процедура Sort вызывает Функцию LessThan и, следовательно, используя процедуру сортировки, мы неизбежно используем LessThan и никакую другую процедуру.

Легко написать процедуру сортировки чисел. Можно также организовать сортировку не по возрастанию, а по убыванию или как-нибудь еще. Для этого в нашей программе нужно изменить только процедуру LessThan. Но как раз это в традиционных языках и невозможно. Если в программе должны быть две слегка разные процедуры сортировки (отличающиеся только функцией LessThan), процедуру Sort все равно приходится дублировать. Видели дома из кирпича? Построили и уже ничего не изменишь - разве что стену долби. А то, что сделано из детского конструктора, легко разбирается и снова собирается.

Можно, конечно, передавать процедуру LessThan как параметр в процедуру Sort. Но в реальной программе таких процедур бывает довольно много. А вызывать процедуру со списком из нескольких десятков параметров - утомительно. Спасти положение может только наследование. Напомним, что объектом называется конструкция, очень похожая на запись, но содержащая, кроме полей данных, еще и поля, содержащие процедуры, работающие с данными этого объекта. Один объект может наследовать, т.е. как бы переносить внутрь своего описания данные и процедуры другого объекта. Для иллюстрации рассмотрим примеры 7 и 8.

Процедуры Sort и Swap описываются в классе СОРТИРУЕМЫЙ, а функция LessThan остается неопределенной - пока не будет описана в классе -наследнике (например, в классе ПОВОЗРАСТАНИЮ). Теперь для каждого нового способа сортировки мы можем описать новый класс - наследника класса СОРТИРУЕМЫЙ, а в нем - только новое определение функции LessThan. И этого будет достаточно. Заметим также, что имя массива исчезло из списков параметров процедур. А именно, теперь пишем так:

```
var
    b: ПоВозрастанию;
...
b.Sort;
```

Эти возможности тем более важны, поскольку при создании программного обеспечения уже давно существует множество проблем. Растут быстродействие и объем памяти вычислительных машин. Растут требования к вычислительной технике, а

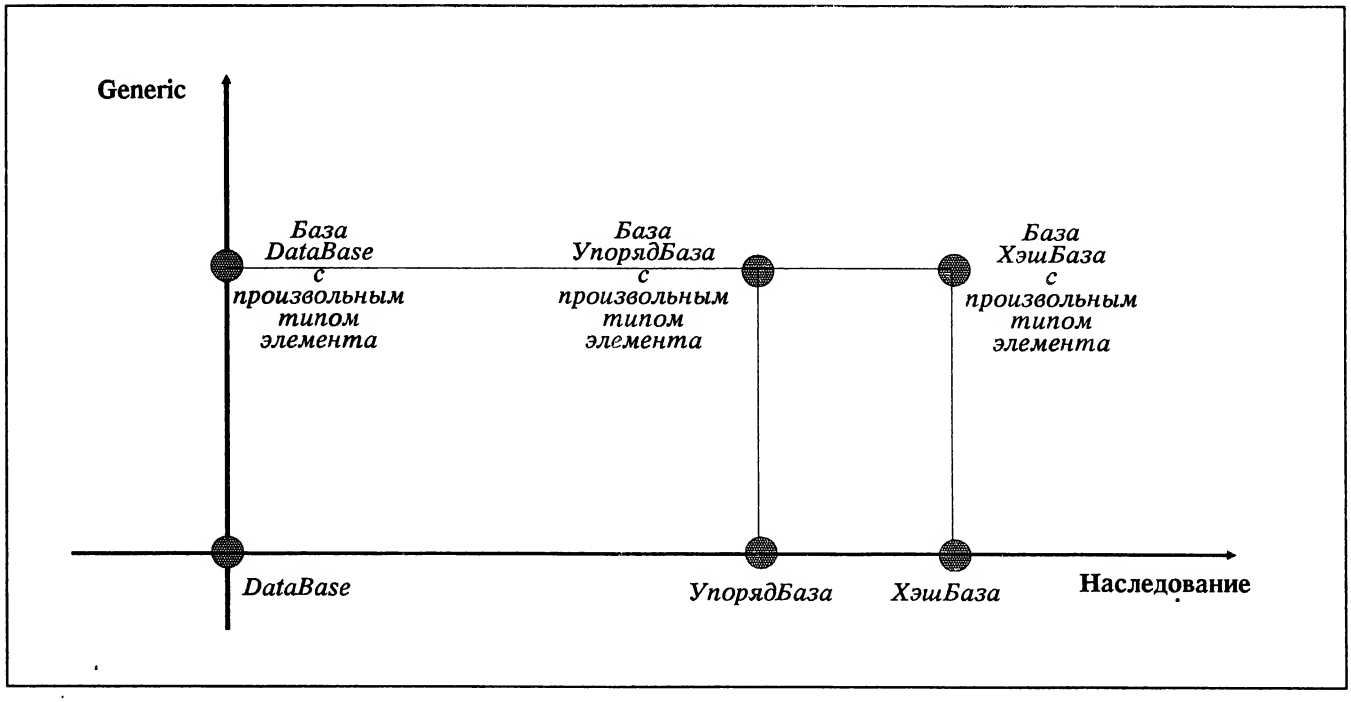


Рис. 3. Возможности совместного использования параметризованных типов и наследования

```
type

  row = array [1..100] of integer;

procedure Sort (var b : row);
begin
  for i:=1 to n do
  for j:=i to n do
    if LessThan(b,i,j)
      then Swap(b,i,j);
end;
```

```
procedure Swap (var b: row; i,j : integer);
var c: integer;
begin
  c := b[i];
  b[i] := b[j];
  b[j] := c;
end;

function LessThan (var b: row; i,j : integer) : boolean;
begin
  LessThan := (b[i] < b[j]);
```

Пример 6. Программа сортировки

значит и к размерам программ. А программирование меняется медленно. И большие программы остаются источником больших ошибок. Программы состоят уже из многих тысяч строк. И только немногие из них являются действительно открытием интеллекта. В программировании существует масса механической работы, когда программист повторяет одни и те же приемы. И идея заставить это делать машину всегда представляется заманчивой. Например, многие процедуры (как и в случае с сортировкой) часто переписываются в нескольких местах программы лишь с незначительными вариациями. Есть надежда, что с помощью объектно-ориентированного программирования от этого удастся избавиться.

Если эти надежды сбудутся, то изменится сам облик профессии программиста. Уже сформировалась новая концепция программирования - под названием "сборочного программирования". Идея состоит в том, что будут писаться модули меньшего размера, но более универсальные. Из них будут составляться постоянно поддерживаемые банки программных компонент; из этих компонент будут собираться программы. Универсальность модулей обеспечит их долгоживучесть и снизит стоимость отдельной программы. Подход уже применялся и оказался удачным. А как широко он будет распространяться, покажет время.

```
type
  row = array [1..100] of integer;

  Сортируемый = object
    b : row;
    procedure Sort;
    procedure Swap (i,j: integer);
    function LessThan (i,j: integer): boolean; virtual;
  end;

  ПоВозрастанию = object (Сортируемый)
    function LessThan (i,j: integer): boolean; virtual;
  end;
```

Пример 7. Организация наследования

```
procedure Сортируемый.Sort (var b: row);
begin
  for i:=1 to n do
  for j:=i to n do
  if LessThan (b, i,j) then Swap (b, i,j);
end;

procedure Сортируемый.Swap (var b: row; i,j : integer);
var c: integer;
begin
  c := b[i];
  b [i] := b[j];
  b [j] := c;
end;

function ПоВозрастанию.LessThan (var b: row; i,j : integer): boolean;
begin
  LessThan := (b[i] < b[j]);
end;
```

Пример 8. Модифицированная сортировка

А.Тихонов

Что–то с памятью моей стало...

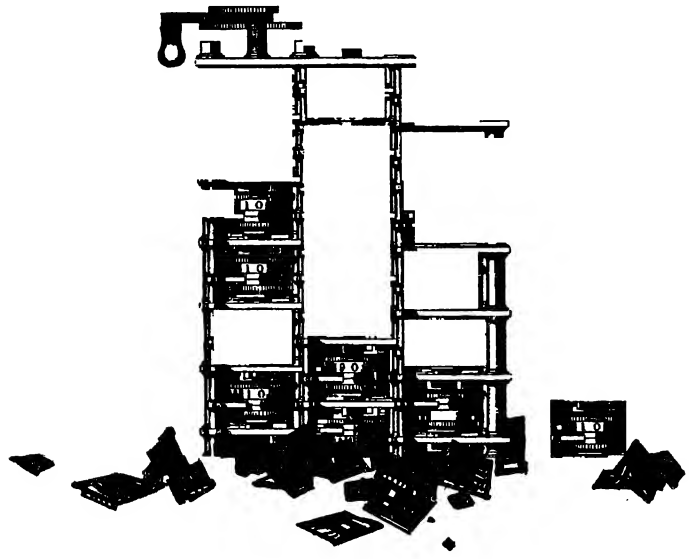
Часть II

В первой части статьи мы проанализировали различные типы оперативной памяти, с которыми приходится сталкиваться пользователю ПЭВМ, совместимых с IBM PC, а также способы ее организации.

Здесь мы попытаемся ответить на вопрос, как на IBM PC работать с большими программами и массивами данных, превышающими 640 Кбайт.

Ответ на этот вопрос многозначный: во-первых, уменьшение требований к ресурсам, используемым операционной системой, во-вторых, использование под управлением ДОС средств, специфичных для микропроцессоров 80286, 80386, и, наконец, разбиение программ и данных так, чтобы загружать с диска в оперативную память необходимые в данный момент сегменты.

Посмотрим сначала, что делают для нас гиганты индустрии программного обеспечения, а затем, чем мы можем помочь себе сами.



Ответ фирмы Digital Research - операционная система Dr.DOS

Статья об этом программном продукте в журнале "Byte" так и называется "Dr.DOS дает надежду" [1]. Создатель Dr.DOS - фирма Digital Research в свое время была законодателем мод в области операционных систем (ОС) для восьмиразрядных микроЭВМ, ею разработано семейство ОС CP/M. Digital Research предложила полностью совместимую с ДОС ОС с большим набором команд и более экономно использующую оперативную память. Если ДОС 3.3 оставляет пользователю не более 530 из 640 Кбайт, то Dr.DOS на стандартной PC/AT с одним мегабайтом оперативной памяти предоставляет пользователю 620 Кбайт, а на ПЭВМ с микропроцессором 80386 в алфавитно-цифровом режиме 704 Кбайта.

Это осуществляется за счет использования первых 64 Кбайт за границей первого мегабайта для загрузки ядра операционной системы, а также неиспользуемых участков адресного пространства между 640 Кбайтами и 1 Мбайтом для загрузки драйверов устройств и резидентных программ.

Последнее существенно при работе в локальной сети и с программами, полностью использующими всю оперативную память ПЭВМ (издательские системы, системы автоматизированного проектирования).

Распределение памяти в операционной системе Dr.DOS представлено на рис.1.

Ответ фирмы QuarterDeck

Пожалуй, первой фирмой, занявшейся увеличением доступной пользователю памяти, была QuarterDeck, основным продуктом которой была многозадачная надстройка над ДОС DesqView.

DesqView работает, в отличие от Windows, в алфавитно-цифровом режиме, позволяет запустить одновременно несколько пользовательских

программ и мгновенно переключаться от одной к другой.

Свои проблемы QuarterDeck решила с помощью драйвера QEMM. Успех этого программного продукта был настолько велик, что QuarterDeck стала продавать его отдельно от DesqView. В комплект поставки входит драйвер для МП 80386 QEMM.SYS, для МП 80286 QRAM.SYS, работающий с дополнительной памятью EMS, а также программы Manifest и Optimize, позволяющие настроиться на конкретное оборудование пользователя.

Загрузка драйверов в память выше 640 Кбайт осуществляется с помощью команды LOADHI. Драйвер VIDRAM освобождает 96 Кбайт памяти, используемой адаптерами EGA и VGA. Например, для загрузки VIDRAM в память и освобождения 96 Кбайт достаточно ввести две команды:

LOADHI VIDRAM

VIDRAM ON

Версия QEMM 5.1 полностью совместима с Windows, но полный набор его возможностей проявляется только для МП 80386. Частично проблемы для МП 80286 решаются с помощью драйвера QRAM.

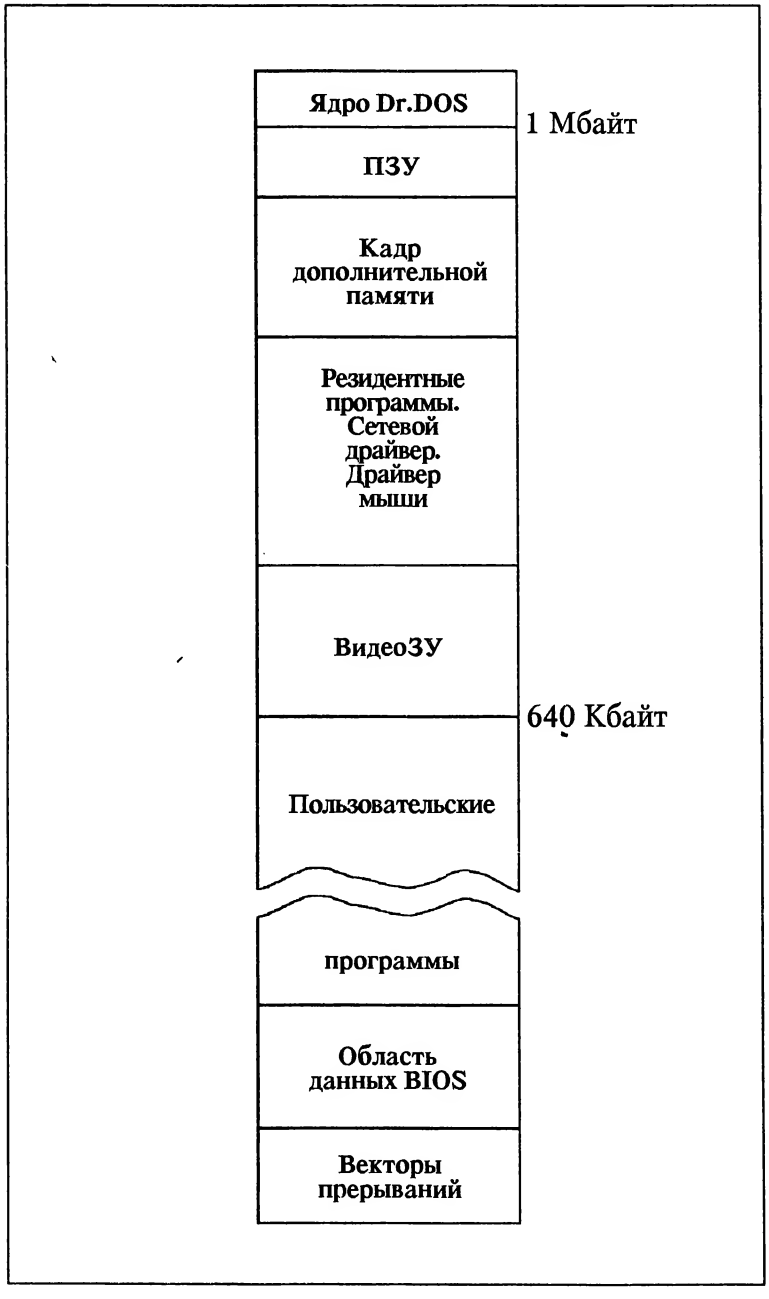


Рис.1. Распределение памяти в операционной системе Dr.DOS

Запоздалая реакция MicroSoft

Выпуск версии ДОС 5.0 в первом полугодии 1991 года является запоздалой реакцией фирмы на рассматриваемые проблемы.

По сравнению с ДОС 4.01 в ДОС 5.0 значительно усовершенствованы средства распределения и управления памятью. Ядро ОС занимает в оперативной памяти 20 Кбайт. Для сравнения укажем, что ДОС 3.3 требовала 55 Кбайт, а ДОС 4.01 67 Кбайт.

Новые возможности ДОС 5.0

В ДОС 5.0 введены дополнительные команды, она работает быстрее, предоставляет удобную графическую командную оболочку, которая однако не заставила меня отказаться от Norton Commander. В комплект поставки входит интерпретатор Quick Basic.

ДОС 5.0 позволяет использовать 64 Кбайта выше 1 Мбайта для работы собственного ядра.

На Бога надейся, а сам не плошай...

Что же может сделать сам программист, чтобы втиснуть свою программу и данные в

640 Кбайт? Сразу отметим, что средства для работы с программами и данными совершенно различны. Рассмотрим сначала работу с данными.

Для ускорения доступа к данным в расширенной памяти может быть создан *виртуальный диск*. Для запуска виртуального диска объемом 300 Кбайт достаточно внести строку

DEVICE=VDISK.SYS 300/E

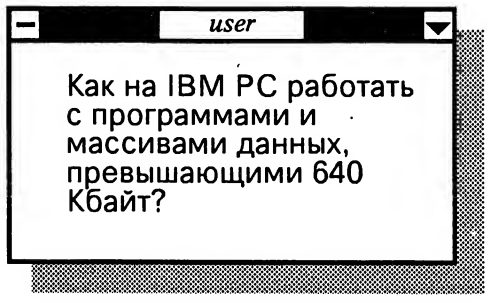
в файл CONFIG.SYS и перезагрузиться. При работе с виртуальным диском необходимо всегда помнить, что при перезагрузке или выключении питания вся информация на виртуальном диске исчезает, поэтому необходимо переписывать его содержимое на твердый диск.

Некоторые программные продукты, например, Turbo Prolog фирмы Borland International, имеют встроенные средства для создания и ведения внешних баз данных в дополнительной памяти. Если Вы решили самостоятельно разработать средства для работы с дополнительной памятью, то рекомендуется воспользоваться библиотекой Turbo Professional для Turbo Pascal. Библиотека содержит легко читаемые исходные тексты программ, которые можно адаптировать для собственных нужд.

Несколько сложнее обстоит дело с программами. Мы уже говорили, что непосредственный выход за пределы магических 640 Кбайт возможен только в защищенном режиме. Такую возможность предоставляют расширители ДОС.

Расширители ДОС

Расширители ДОС (DOS Extender) представляют собой надстройки над ДОС, позволяющие после запуска пользовательской программы переходить в защищенный режим и работать со всей доступной памятью. Для обработки прерываний и функций ДОС расширитель вынужден возвращаться в реальный режим. Данная операция для МП 80286 занимает достаточно много времени, при переходе из режима в режим может быть потеряно прерывание, что в свою очередь может привести к зависанию системы.



Как создать оверлейную программу

Расширители ДОС обычно используются, когда необходим перенос на ПЭВМ больших программных продуктов, разработанных для других типов ЭВМ. Примером расширителя является DOS/16M фирмы Rational Systems Inc. DOS/16M работает с МП 80286 и 80386, но не поддерживает 32-разрядной адресации МП 80386. Полностью возможности 80386 использует расширитель 386/DOS фирмы PharLap. В качестве расширителя ДОС может использоваться и Windows 3.0.

Следует отметить, что для работы в защищенном режиме необходимо применение специальных трансляторов для языков высокого уровня, таких как C, Fortran, Pascal.

Использование расширителей ДОС представляет собой достаточно экзотическую затею. Теперь мы обратимся к более традиционным методам снижения требований к объему оперативной памяти, самым распространенным из которых является создание программных модулей с перекрытиями.

Оверлеи

Использование структур программных модулей с перекрытиями (оверлеев) базируется на том, что при запуске программы в оперативной памяти постоянно находится только небольшая часть ее, которая управляет загрузкой, выполнением и освобождением памяти для остальных программных модулей. Построение программных модулей с перекрытиями осуществляют редакторы связей, такие, как LINK, Plink86, RTLink.

Поясним процесс создания оверлейной программы на примере. Пусть программа P (рис.2) состоит из объектных модулей A, B, C, D, B1, B2, D1,D2. Головной модуль A вызывает все остальные модули и должен быть резидентным (постоянно присутствовать в оперативной памяти). Модули B, C, D не зависят друг от друга и могут сменять друг друга в одной области оперативной памяти. *Областью* называется участок оперативной памяти, в который осуществляется поочередная загрузка модулей. Размер области определяется размером самого большого модуля, загружаемого в нее. Размер первой области определяется размером модуля B (рис.3).

Во второй области друг друга сменяют модули B1, B2, D1, D2. Редактор связей LINK помещает все оверлеи в выполняемый .EXE модуль; PLINK86 и RTLINK позволяют наряду с этим помещать оверлеи в отдельные файлы. Это бывает удобно, если необходимо работать с программой размером более 360K на ПЭВМ без твердого диска. В этом случае оверлейные модули могут быть распределены по дискетам.

Построение программ с перекрытиями требует от программиста аккуратности, особенно для программ с числом модулей, превышающим сто, так как некорректное построение структуры с перекрытиями приводит к зависанию системы.

Загрузка осуществляется *оверлейным менеджером*, входящим либо в библиотеку соответствующего компилятора, либо в библиотеку редактора связей.

Своеобразная структура объектных модулей Turbo Pascal не позволяет воспользоваться стандартным редактором связей и заставляет компилятор брать на себя работу по созданию программных структур с перекрытиями. Turbo Pascal позволяет управлять объемом оперативной памяти для оверлеев, использовать для работы с ними дополнительную память, а в [2] показано, как для той же цели использовать расширенную память.

Виртуальная память

Более изощренную технику для построения больших программных систем предлагает фирма Borland. Называется она *виртуальным объектно-ориентированным менеджером памяти реального времени* (VROOMM) и реализуется компилятором Turbo C++. Кроме того, фирма интенсивно использует эту технику для собственных разработок Reflex 2.0, Quattro Pro, Paradox.

Принцип действия VROOMM заключается в разбиении программного кода на участки размером от 2 до 4 килобайт, называемые *объектами*. Иногда размер объекта может достигать 16 Кбайт [3].

Объекты грузятся в память по мере необходимости, при этом выгружаются уже отработавшие объекты.

В отличие от оверлеев, практически вся работа по разбиению программы на объекты выполняется компилятором, а не редактором связей. VROOMM позволяет экономить память по сравнению с оверлеями и, что немаловажно, автоматизирует построение программы.

Еще одной интересной особенностью VROOMM является возможность использовать для хранения вытесненных из оперативной памяти объектов дополнительную память. При этом вытесненный объект в случае необходимости будет загружен из дополнительной памяти, а не с диска, что значительно повышает быстродействие программы.

Аналогичная техника используется в системе Clipper 5.0 при редактировании связей с помощью редактора связей RTLink. При этом разбиению подвергаются программные модули Clipper, однако еще в этом году предполагается выпуск версии RTLink, позволяющей использовать виртуальную память для программ, транслированных с помощью компилятора MicroSoft C.

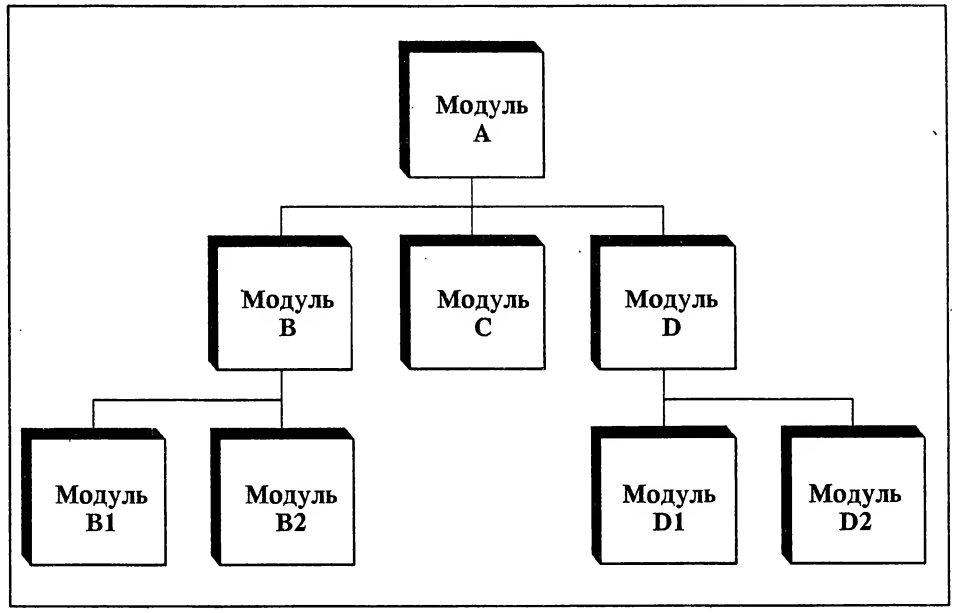


Рис. 2. Дерево вызовов модулей для программы с перекрытиями

Динамическая загрузка

Очень простую, но менее эффективную возможность экономии оперативной памяти дает функция ДОС Exec (4Bh), позволяющая из программы загрузить и выполнить .EXE или .COM - модуль [4]. Данная функция присутствует практически во всех системах программирования. В C она называется spawn, в Turbo Pascal - exec, а в Turbo Prolog - system. Exec позволяет передать вызываемой программе строку, а вот получить информацию из вызываемой программы можно только через файловую систему.

Обычно для экономии в оперативную память загружается небольшой программный модуль, выполняющий только функции загрузки и передачи управления для других выполняемых модулей.

При загрузке часть кода загружающей программы может быть временно вытеснена на диск или в дополнительную память. Исходные тексты средств вытеснения для Turbo Pascal

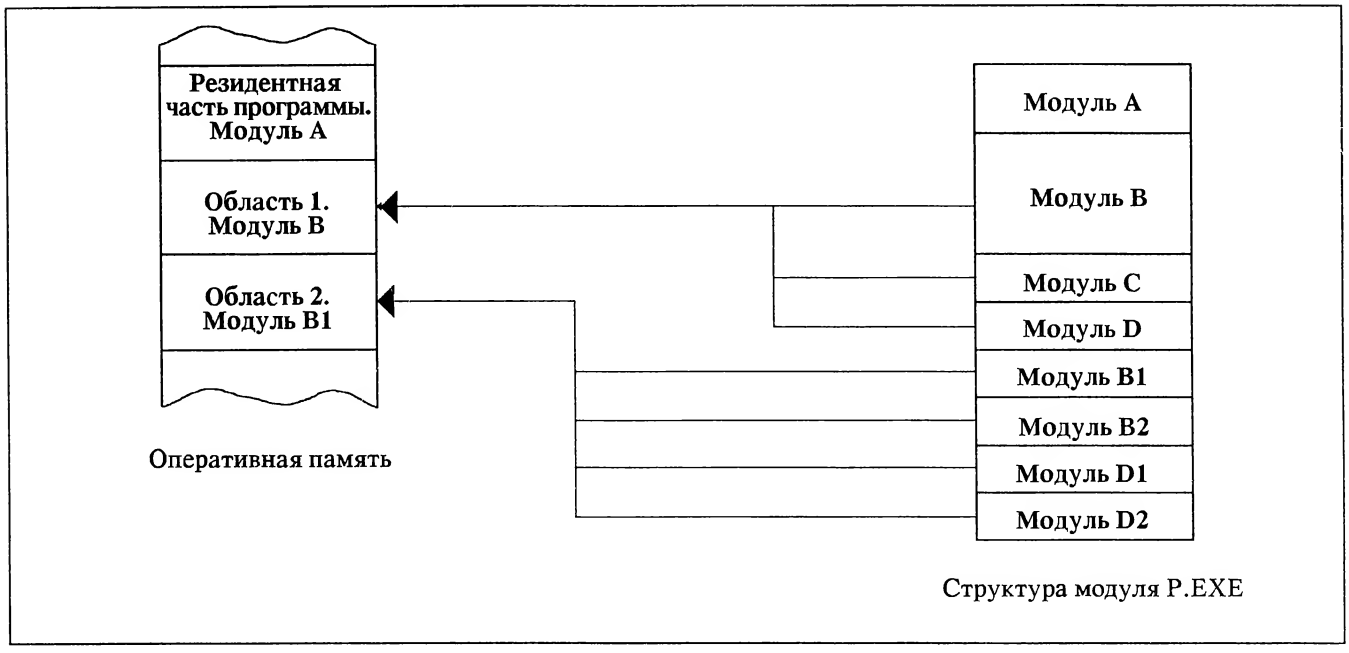


Рис. 3. Загрузка оверлеев в оперативную память

имеются в библиотеке Turbo Professional.

Что же дальше?

"Человек сильнее", - обычно отвечаю я, когда мне говорят, что катастрофически не хватает машинного времени и памяти. Действительно, иногда через час, а иногда и через неделю выход находится, и он обычно связан с переформулировкой задачи, поиском новых методов решения или с тем и с другим.

Но есть большое число задач, с которыми этот трюк не проходит. К ним относятся обработка трехмерных изображений, работа с большими распределенными базами данных, работа в системах реального времени, задачи искусственного интеллекта. И это естественно. IBM PC задумывалась и реализовывалась как однопользовательская и однозадачная ПЭВМ для деловых приложений.

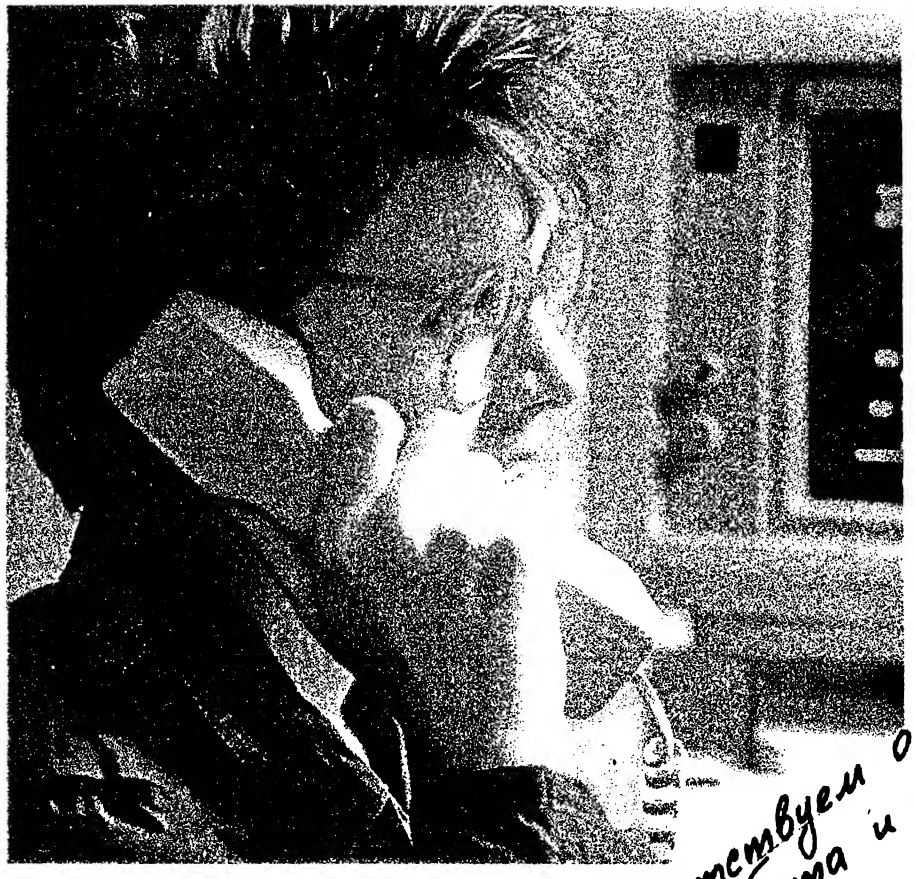
Я очень люблю ПЭВМ, позволяющую мне работать, когда у меня есть желание и настроение, быстро отладить программу, увидеть на экране результаты своей работы в графической форме, не зависеть от машинисток и администрации ВЦ.

Вычислительная мощность ПЭВМ растет, появляются новые программные средства, расширяющие их функциональные возможности. Предполагается, что новые версии Windows и трансляторов для языков программирования высокого уровня позволят легко работать в защищенном режиме со всеми доступными ресурсами оперативной памяти и, что самое важное, создавать прикладные программы, работающие в средах ДОС, OS/2 и даже UNIX, за счет унификации системных вызовов.

Но я также помню, что рабочие станции, работающие под управлением операционной системы Unix, лучше чем ПЭВМ работают в локальных сетях и с интенсивными графическими приложениями, а ЕС ЭВМ, которую сейчас никто не любит, с большими базами данных. Когда передо мной ставится новая задача, прежде чем начинать стучать по клавишам, я думаю, а можно ли ее решить на ПЭВМ, к чему призываю и Вас.

Литература

1. Wood R. Dr.DOS offers hope. Byte, 1990, v.15, N'12.
2. Grober M. Overlays verladen. C't magazine fur computer technik, 1990, N'1.
3. Bertolucci J. The 640K-byte solution? Byte, 1990, v.15, N'3.
4. Чижов А.А. Системные программные средства ПЭВМ. Справочник. М.: Финансы и статистика, СП ПараГраф, 1990.



Приветствуем отделы сбыта и снабжения!

Если у Вас есть проблемы со сбытом или снабжением, сегодня же позвоните в

"ГРАНИТ"

Малое предприятие "ГРАНИТ" - это:

- брокерская контора No.869 на Российской товарно - сырьевой бирже;

- представительство на Российской агропромышленной бирже,Владивостокской бирже "Азия", Уральской бирже, Казахстанской республиканской бирже;

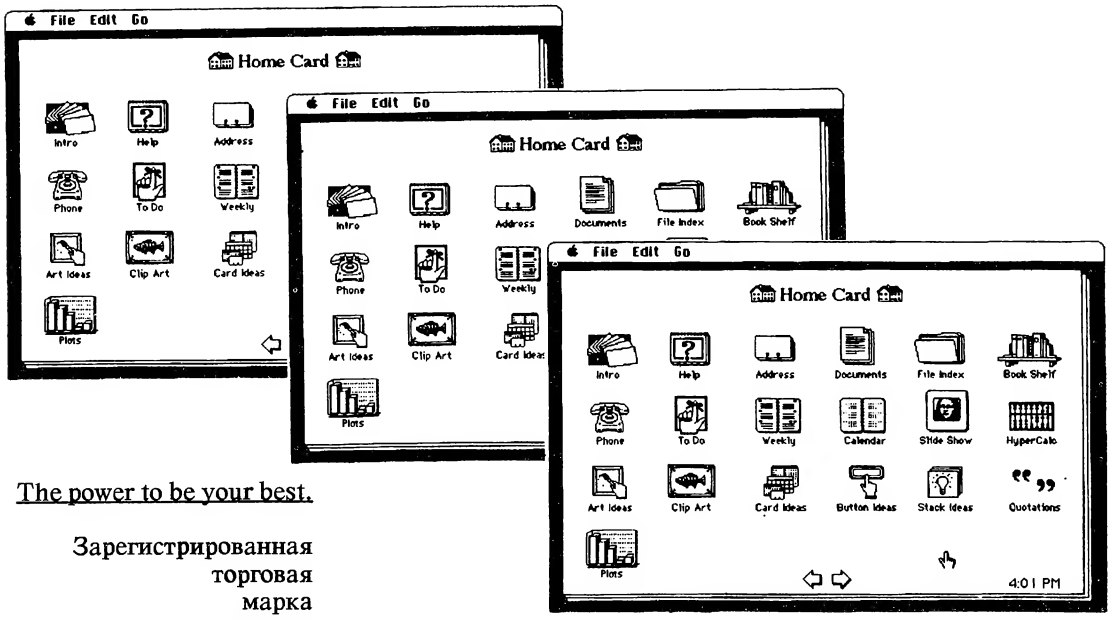
- связи с десятками предприятий и организаций во всех регионах страны.

ПОМНИТЕ: "ГРАНИТ" ЖДЕТ ВАШЕГО ЗВОНКА!

Тел.(095) 236-03-69, 237-11-45.

Куда катится компьютерное яблоко

В.Крюков, С.Пучков



The power to be your best.

Зарегистрированная торговая марка фирмы Apple Computer.

Введение

Компьютер обрабатывает информацию. Как Вы работаете с компьютером, зависит от того, каким образом Вы предоставляете ему информацию и как получаете ее обратно. Это взаимодействие на компьютерном жаргоне называется интерфейсом между человеком и компьютером.

Большинство из нас, по-видимому, желает, чтобы компьютер работал так же, как мы со знакомыми нам образами и концепциями.

Когда вы включаете компьютер с традиционным интерфейсом, на дисплее появляется некое загадочное C:>, и если Вы не знаете, что делать дальше, Вам придется изучить довольно объемистое руководство. Тысячи людей прилагают усилия, чтобы овладеть этим искусством, и нельзя сказать, что это невозможно. Но этот процесс отнимает много сил и времени. В этой статье мы постараемся познакомить Вас с малораспространенным в нашей стране персональным компьютером /ПК/ Macintosh фирмы Apple. Немногочисленные публикации об этом компьютере касались в основном аппаратного уровня [1,2,3]. Здесь мы попытались проанализировать особенности уникального программного обеспечения Macintosh.

С одной стороны, у этого вопроса есть замечательная история со своей интригой и развитием. Эта история может показать, за счет какой стратегии компания Apple со своим необыкновенным ПК Macintosh долгое время не только удерживалась на передовых позициях компьютеризации, но и вела активное наступление. С другой стороны, это поможет выявить генеральную концепцию развития и приоритеты фирмы Apple в области компьютеризации.

Для иллюстрации материала статьи нами выбрана область компьютеризации образования, как представляющая несомненный интерес для большого круга читателей, которые, по-видимому, обречены всю жизнь учиться в нашем быстро меняющемся мире.

Для иллюстрации широкого применения персонального

компьютера Macintosh фирмы Apple в образовании, было бы достаточно привести несколько впечатляющих данных: например, 65% ПК, приобретенных американскими школами в 1989 году, произведены фирмой Apple [4]. Для сравнения у IBM - 26%. Что же еще здесь добавить?

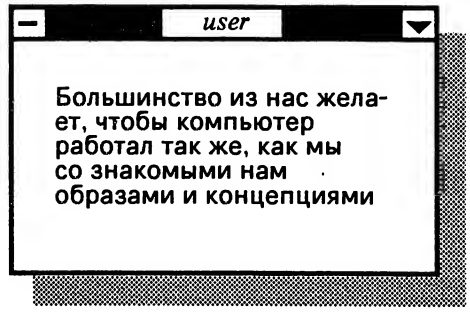
Интерпретация эволюции ООП в программном обеспечении ПК Macintosh

Сразу же следует указать, что идеология программного обеспечения ПК Macintosh является полностью оригинальной и неповторимой. Корни этой идеологии уходят в плодородную почву объектно-ориентированного программирования (ООП).

Действие первое

В 70-е годы в научном центре Palo Alto фирмы Xerox Группа Исследования Образования вела разработку компьютерной системы, которая была бы доступной для работы неподготовленному человеку и раскрепощала бы творческий потенциал пользователя. В результате появился новый язык-система программирования Smalltalk-80, который, как казалось в начале 80-х годов произведет революционный переворот в компьютеризации. Действительно при личном знакомстве со Smalltalk на практике, он производит очень приятное впечатление. Однако уже скоро многим стало ясно, что Smalltalk останется академически классическим объктно-ориентированным языком, но не будет широко использоваться на практике. Этому мешает ряд причин фундаментального характера.

1) Smalltalk-80 реализует слишком чистую объектно-ориентированную концепцию с претензией на создание абсолютно всепокрывающей среды, владеющей всеми ресурсами ЭВМ и объектами. Это привело к неразрешимости проблемы обмена объектами между пользователями и невозможности скооперироваться [5].



2) Smalltalk-80 так и не стал рабочим инструментом непрофессионалов, когда программирование ведется на квазиязыке предметной области в виде как бы игры в кубики, так как остается неустранимым момент написания процедурной части [6].

SmallTalk открыл новое направление в объектно-ориентированной концепции

РЕЗЮМЕ. Итак, Smalltalk открыл новое направление в компьютеризации и вызвал волну энтузиазма и инициативных разработок по мотивам объектно-ориентированной концепции, но эта волна разбилась о рифы практики, и только там, где были возможности для прохода, потекли плодотворные идеи.

Одним из таких открытых для движений направлений явилась разработка ПК Macintosh под руководством Стива Джобса.

Действие второе

В начале 80-х годов Джобс сделал визит в научный центр Palo Alto и познакомился с изобретениями Группы Исследования Обучения. Это определило направление работ по созданию нового компьютера фирмы Apple Macintosh. Действительно, глядя на окна на экране ПК Macintosh и на работу с мышкой, нельзя не признать его родство со Smalltalk.

Все аналитики в один голос отмечали высокую эффективность нового графического интерфейса ПК Macintosh с пользователем, подсчитывали повышение производительности труда. И это действительно так: согласно исследованиям фирмы Diagnostik Research Inc., проведенным в 1990 г., продуктивность при работе на Macintosh на 37% выше, чем на персональных компьютерах под МС ДОС [7].

Иногда специалисты указывали на наличие объектно-ориентированного подхода в интерфейсе ПК Macintosh. "Объектно-ориентированный подход программ наиболее точно соответствует характеру проектировочных работ, повышает общую производительность системы человек-машина (соотношение выход работы/затраты времени на 20-40% выше, чем на станциях IBM PC)" [8].

Для уточнения, что составляет суть объектно-ориентированного подхода, рассмотрим его признаки на примере оболочки операционной системы. Корень файловой системы, например, изображается в виде папки с документами. При открытии этой папки на экране появляются пиктограммы документов-файлов.

Манипулируя мышью и клавиатурой, Вы можете располагать пиктограммы файлов в любом месте экрана (окна), можете рассортировать их по функциональному назначению в папки с помощью программного средства folder, копировать содержание целиком или частично и выполнять многие другие операции точно так же, как вы привыкли делать это за письменным столом с рабочими документами.

Для того чтобы запустить

программу, необходимо указать маркером в виде стрелки на нужную пиктограмму и нажать кнопку.

Здесь мы имеем дело с объектно-ориентированным интерфейсом, однако *остается не до конца выясненным вопрос, откуда появляется дополнительная мощность при работе в таких условиях.*

Если вернуться к вопросу об ООП, то можно сказать, что его сущностью является новый, а на самом деле старый, способ структурирования (декомпозиции) задачи, который знаком и привычен нам по жизненному опыту. В жизни большинство людей имеет дело с объектами и поэтому думает в терминах объектов, наделенных теми или иными функциями, которые эти объекты выполняют. Традиционное же программирование предлагает думать наоборот - в терминах абстрактных функций, которые выполняют обработку объектов (информации).

Среда объектно-ориентированного программирования несет в себе противоположную от традиционной парадигму, или взгляд на суть предмета, который более адекватен способу мышления и приемам решения задач прикладных инженерных областей [6]. Это положение и является источником роста производительности труда при работе за ПК Macintosh. Еще более этот момент важен для работы детей за компьютером, когда ребенок, оставаясь в родной квазиреальной среде, получает информацию (знания) в привычном естественном виде.

РЕЗЮМЕ. Итак, прагматичный Стив Джобс отсек проблемы Smalltalk, поместив объектно-ориентированную среду в верхнем уровне программного обеспечения ПК Macintosh - в оболочке операционной системы. Этого оказалось достаточно, чтобы существенно улучшить стиль взаимодействия с ЭВМ. Существует тенденция переноса объектно-ориентированного подхода на более низкий уровень: чтобы сделать пользователя независимым от программиста.

Гипертекстовый продукт HyperCard фирмы Apple стал платным системным средством

Поэтому логично было бы ожидать попыток продвинуть объектно-ориентированный подход на более низкий уровень программного обеспечения ПК Macintosh.

Действие третье и не последнее

И вот, когда в 1987 г. фирма Apple объявила, что поставляет свой гипертекстовый продукт HyperCard бесплатно, т.е. он фактически стал штатным системным средством, стало ясно, что это исторический шаг в том же направлении.

Так что же такое гипертекст?

Со времени изобретения наборных шрифтов книги и журналы существенно не изменились: текст и рисунки организуются последовательно, одна тема следует за другой. Это противоречит тому факту, что наш мозг помнит и хранит информацию в некотором смысле способом, структурно более похожим на паутину, чем на прямую линию.

Еще 40 лет тому назад культурологи, такие как Ванневер Буш, научный советник президента Рузвельта, предсказывал способность быстро просматривать электронным образом текст большого объема, связывающий родственные идеи, следуя особыми путями, которые прямо отражают наши потребности или интересы и позволяют проходить мимо не относящейся к делу информации.

Термин "гипермедиа" используется для обозначения способности связывания не только слова, но еще и графических образов, звука и видеоизображения

В 60-х годах Тед Нельсон, другой компьютерный провидец, ввел в обиход термин гипертекст, имея в виду мощь и гибкость такой среды. Поскольку концепция гипертекста была также распространена на другие средства передачи информации, термины "гипермедиа" и "интерактивная мультимедиа" были использованы для обозначения способности связывать не только слова, но еще графические образы, звук и видеоизображение. Сегодня гипертекст и интерактивные мультимедиа становятся реальностью через развитие продуктов типа HyperCard.

Apple ввела в употребление HyperCard в августе 1987 года. Хотя существовал и не единственный продукт такого типа, HyperCard достиг гораздо большего распространения и признания, чем его конкуренты, частично благодаря торговой стратегии Apple: компания снабжала бесплатной копией каждый вновь приобретенный компьютер Macintosh. Доступность HyperCard для такого большого количества пользователей породила целую индустрию прикладных систем HyperCard за низкую цену, что способствовало дальнейшему росту популярности программного продукта.

Что конкретно может HyperCard

HyperCard и другие программы, основанные на принципе гипертекста, устанавливают новый жанр программного обеспечения, который делает возможным представление знаний и информации так, как делаем мы в наших мыслях - непоследовательно, при помощи ассоциаций и контекста.

HyperCard компонуется из карт совсем как картотечный ящик индексных карточек. Комплект карточек формирует стек. Каждая карточка содержит одну или несколько кнопок, активизируемых мышью. С помощью кнопок можно также выполнять вычисления и даже управлять периферией такой, как плейер видеодиска.

Вы можете разработать свой собственный стек: для этого необходимо создать карточки, впечатать в них текст, сконструировать иллюстрацию с помощью инструментария для рисования, вставить графику

из других прикладных систем и разместить кнопки на картах для их связывания. Вы можете управлять любым объектом (кнопкой, полем, карточкой или стеком) при помощи написания на англоподобном языке программирования HyperTalk сценария (script) для определения поведения объекта. (Система HyperCard действительно является объектно-ориентированной системой: она имеет ограниченное количество объектов (карточки, поля, кнопки), с которыми можно производить какие-либо естественные манипуляции).

Рассмотрим более подробно HyperCard. Логически можно выделить пять уровней работы с системой.

1) Уровень "Browsing" - самый простой режим работы в среде, позволяет просматривать уже существующие стеки, исследовать логические связи с помощью меню "Go" или кнопок на карточках. На этом уровне вы не можете делать каких-либо изменений в информационном стеке.

2) Уровень "Typing" позволяет впечатать текст в информационное поле, делать вставки, копировать рисунки, точно также, как вы делаете это в редакторе текстов.

3) Уровень "Painting" дает вам доступ к инструментарию HyperCard, позволяет менять дизайн карточек. Функционирование стеков, т.е. то, что происходит, когда вы активизируете кнопку на карточке, не может быть изменено на этом уровне.

4) Уровень "Authoring" позволяет создавать собственные поля и кнопки, и с помощью "link to" вы можете заставить HyperCard действительно писать сценарий за вас.

5) Уровень "Scripting" - самый сложный профессиональный режим работы: написание сценариев на англоподобном языке прграммирования HyperTalk.

Язык HyperTalk хотя и является по мощности универсальным языком программирования, но имеет четко очерченную область применения, и поэтому легко изучается в условиях постепенного погружения в систему и естественного повышения уровня работы на фоне решения практических задач. HyperTalk привлекает много внимания со стороны непрограммистов, в особенности связанных с учебным процессом - преподавателей начальных и средних школ и колледжей. Это объясняется тем, что инструменты языка HyperTalk делают для начинающих относительно простым достижение впечатляющих результатов.

HyperTalk привлекает много внимания со стороны непрограммистов, в особенности связанных с учебным процессом

Хотя создание оригинальных стеков все еще отнимает значительное количество времени и требует планирования, процесс легче, чем в других существующих системах для творчества. Еще легче модифицирование любого из растущего числа коммерческих и авторских стеков.

Преподаватель как менеджер информации

Рассмотрим вопрос, каким образом мультимедиа может быть полезен учителю в классе.

Представьте, что вы ведете в школе учебные часы по истории средних веков. Представьте уже совсем невероятные для нашей действительности вещи, что при помощи построения стека в HyperCard на ПК Macintosh Вы создали текстовые карточки, относящиеся к повседневной жизни, описывающие законы и правительства и обсуждающие роль религии в средние века. Вы сканировали рисунки для иллюстрации архитектуры и искусства, одежды этого периода, карты и диаграммы. Вам доступен CD-ROM и плейер видеодиска с соответствующим музыкальным и видеосопровождением. В конце концов для ведения уроков в классе по определенным темам Вы подготовили средства мультимедиа.

Используя HyperCard вместо ручки и бумаги, студенты могут сконструировать оригинальную программу

Итак, Вы ведете урок и в середине вашего рассказа о распространении христианства на Руси, ученица поднимает руку и спрашивает о взаимоотношении с язычниками в те времена. Используя в HyperCard свойство "find" или "go", вы локализуете связанные карточки из темы киевская Русь (язычество древних славян), летописание древней Руси ("Повесть временных лет","Слово о полку Игореве"), жизнь и деяния князя Святослава (военные походы). Позже Вы осуществите доступ к видеозаписям фильмов "Андрей Рублев" и "Александр Невский". Когда вы продолжите тему, то сможете импровизировать и углубить содержание ответа на заданный вопрос. А когда прозвенит звонок, вы можете "заморозить сегодняшнюю дискуссию так, чтобы, в следующий раз вернуться туда, где Вы прервались.

Таким образом, *Вы как учитель располагаете гибким оперативным инструментом, который может успешно применяться для ведения урока в классе.*

Студенты как авторы

Кроме того, что средства мультимедиа успешно могут применяться учителем, они могут быть весьма эффективны при самостоятельной работе ученика. Этот факт установлен американскими учителями.

Несмотря на то что в США разработан ряд коммерческих и любительских стеков, многие преподаватели утверждают, что *наибольшее значение HyperCard обнаруживается при использовании его* студентами в *качестве творческого инструмента.* Используя HyperCard вместо ручки и бумаги, студенты могут сконструировать оригинальную программу (презентацию) или урок для своих одноклассников. Научные доклады (по географии, биологии и т.д.) полу-

чают новую жизнь при помощи карт, диаграмм и мультипликаций. Студенты проявляют необычный интерес к процессу написания, когда предоставляется случай интерактивного сочинения на базе HyperCard.

HyperCard дает молодым людям возможность быть распорядителями и распространителями информации. Когда они конструируют свои собственные стеки, они вынуждены решать вопрос не только, какую информацию излагать, но и как излагать: рисунком, фотографией, видеозаписью. Через концентрацию внимания такого рода они учатся видеть тему по новому, более тонко.

Calendar

Таким образом, налицо ситуация, когда намечается массовый прорыв в программирование непрофессионалов-пользователей, получивших мощный инструмент создания компьютерных информационных сред. (Аналитиками признается, что HyperCard явился ключом успеха ПК Macintosh на рынке мультимедиа и интерактивного обучения [9]).

РЕЗЮМЕ. Итак, создание системы HyperCard представляет собой событие проникновения объектно-ориентированного подхода из оболочки операционной системы на прикладной уровень. Фирма Apple, сделав HyperCard стандартной принадлежностью ПК Macintosh, открыла дорогу к популярной универсальной интегрированной информационной среде.

Book Shelf

Обобщение рассуждений

Появление языка-системы Smalltalk-80 на рубеже 70-80-ых годов выявило существование объектно-ориентированного направления в развитии программирования, перпендикулярного традиционному. Новое направление является "дружественным" по отношению к пользователю. Традиционное направление в это время идейно "выдыхалось" и находилось в тупике. На базе нового подхода был построен интерфейс пользователя с компьютером Macintosh фирмы Apple. Затем в 1987 году был сделан следующий шаг в этом направлении - введение в широкое употребление системы HyperCard. На этом очертания перспектив размыты, но достигнутый рубеж приводит к осознанию неожиданной мысли - так как работа человека с гипертекстовой системой объединяет в себе два аспекта:

- быстрое и эффективное получение информации и
- владение элементами программирования,

то лозунг "Программирование - вторая грамотность" в этом контексте становится точным и конкретным. До сих пор он принимался интуитивно на веру. Таким образом, все сходится и можно считать, что определено генеральное направление в ранее очерченной нами области. Это учебные системы на базе гипертекста и мультимедиа.

Теперь необходимо подвести генеральную черту в заявленной нами в начале статьи теме. Можно уверенно утверждать, что фирма Apple обладает первоклассным продуктом с передовой идеологией, ориентированной на пользователя. (Это, конечно, нравится покупателям.) ПК Macintosh как продукт несет в себе целостную концепцию, которая сама по себе в определенном смысле формирует философию развития. Это положение только подтверждает мысль, что для успеха на рынке, в том числе и на компьютерном, необходимо располагать современным продуктом, при создании которого исходили из проблемы реальной помощи пользователю в решении его задач.

Литература

1. Д.Хейд, Macintosh против PC (Критическое сравнение семейств ПК), "МИР ПК", 1991, № 2.
2. П.С.Райман, Новые ветви на старой яблоне: Mac Classic, Mac LC, Mac IIsi, "Мир ПК", 1991, № 2.
3. А.И.Адамович, В.А.Крюков, А.И.Шальков. О рынке персональных компьютеров Macintosh и PS/2, "Интеркомпьютер", 1991, №№ 1-2.
4. Classroom Computer Learning, v.10, N 8.
5. В.А.Крюков, Актуальная проблема объектно-ориентированного программирования. Тезисы докладов Всесоюзной конференции по проблемам разработки и внедрения экспертных систем, Москва, 1989, июнь.
6. В.А.Крюков, Анализ принципов объектно-ориентированного программирования, "Микропроцессорные средства и системы", 1989, № 4.
7. PC Week, 1990, № 41.
8. Фирменные материалы фирмы Comdata Microcomputer.
9. N.Baron, Apple deals a new HyperCard, "Byte", July, 1990.

Господа программисты!

Вы талантливы? Ваши идеи интересны?

Об этом должны знать коллеги и конкуренты.

Помните!

Ваши статьи могут появиться на страницах "Монитора"!

Оперативность публикации и высокие авторские гонорары гарантируются.

Адрес редакции: 117419, Москва, а/я 765, "Монитор"

МОНИТОР

МОНИТОР

МОНИТОР

"СОФТ-МОСКВА" приглашает

к сотрудничеству по распространению печатной продукции (журнал "Монитор", книги для программистов) организации и частных лиц

торговая скидка до 20 %

телефон для контактов: 126–31–56

Язык Рефал – первое знакомство

И. Скрипачев

Многим программистам - как любителям, так и профессионалам, приходилось в своей работе сталкиваться с задачей обработки текстов, вообще символьной информации. В процессе работы над программой появляются какие-то индексы, ссылки, которые имеют мало отношения к тексту (не считая головоломных проблем при попытке отладить эту программу) и через некоторое время выясняется, что автор программы забыл где-то увеличить ссылку или "промахнулся" на один символ. Каждый, кто хоть раз писал программы форматирования текста или преобразования текстовых файлов, может привести свои примеры. То же самое можно сказать о программах, выполняющих статистическую обработку строк, текстов и т. д.

Вместе с тем такие программы обладают определенной спецификой, выраженной в том, что программист хорошо представляет конечный результат еще до написания самой программы. Например, результат работы процедуры удаления лишних пробелов в тексте очевиден. Данные такими программами обрабатываются в основном последовательно. Даже если это не так, то в любом случае в каждый момент времени сначала выделяется некоторый контекст (обычно небольшой), который на данном этапе и будет обрабатываться.

Под контекстом неформально будем понимать совокупность (группу) логически связанных символов, обрабатываемых единовременно. Например, для программы сжатия текстового файла по словам таким контекстом будет слово; в программе, разделяющей слово на слоги, контекстом будет слог - одна гласная с несколькими окружающими ее согласными и т. д. Важно отметить, что в качестве контекста берется, во-первых, минимально возможная (по количеству) группа символов, а во-вторых, замкнутая в себе группа, которую можно вычленить из текста (т.е. группа мало связанная или вообще не связанная с остальными контекстами). Для Рефала понятие контекста является одним из основополагающих, поскольку любая программа, написанная на нем, работает с контекстами.

В языке программирования (ЯП), подобном Паскалю, контекст выделяют с помощью последовательности операторов или вызовов соответствующих процедур. Это неудобно, особенно для больших контекстов, потому что, во-первых, программист должен "держать" в голове все строение этого контекста, а во-вторых, перед тем, как с ним работать, нужно предусмотреть в программе какие-то операторы для его распознавания. В программе строение контекста и частично его использование в явном виде не

указаны. В неявном виде оно, конечно, присутствует - это последовательность операторов, обрабатывающих данный контекст. Этот аспект и создает сложности при написании и восприятии программ. Если бы удалось "встроить" контекст в программу в явном виде, то труд программиста значительно бы облегчился, да и понимать программу стало бы проще. В принципе, если в язык программирования ввести конструкцию, позволяющую это делать, то фрагмент программы, который "ищет" данный контекст, становится излишним. Обработка же контекста в общем случае состоит в замене его другим. Поэтому, если в языке программирования предусмотреть операцию такой замены, то становится излишним вообще весь прежний фрагмент программы.

```
program Reverse;

type
  str = string[80];

var
  s : str;
  c : char;
  i,n : integer;

begin
  s:='abcde';
  n:=strlen(s);
  for i:=1 to n div 2 do
  begin
    c:=s[i];
    s[i]:=s[n-i+1];
    s[n-i+1]:=c
  end;
  writeln(s)
end.
```

```
program Reverse;

type
  str = string[80];

var
  in,out : str;
  i,n : integer;

begin
  in:='abcde';
  out:='';
  n:=strlen(in);
  for i:=1 to n do
    out:=in[i]+out;
  writeln(out)
end.
```

Пример 1. Две программы, "переворачивающие" строку

Интересно, что если в ЯП присутствуют эти две конструкции, то прочие операторы (в частности, цикл, ветвление, присваивание) становятся ненужными. Программа на таком языке превращается в совокупность правил замены одних контекстов на другие. Остается проблема выбора надлежащего правила, но она не представляет труда, так как можно задать порядок правил и варианты перехода от правила к правилу, например, их последовательный перебор. Эта проблема касается языка, а не программы.

Такой ЯП, конечно, не претендует на универсальность. Более того, он будет узко специализированным языком - например, написать на нем достаточно сложный вычислительный алгоритм будет проблематично. Но для задач символьной обработки он вполне подходит. Таким языком является Рефал.

Рассмотрим простой пример: программа, "переворачивающая" строку. Это очень простая программа, но даже на ее примере хорошо видны некоторые выигрышные стороны Рефала в сравнении с классическими ЯП. Две нижеприведенных программы на Паскале (пример 1) решают эту задачу немного различными способами: левая работает с одним массивом символов, попарно переставляя местами его элементы, правая переписывает символы в обратном порядке из входного массива в выходной, при этом знак "+" обозначает сложение (сцепление) строк.

Программа FIRST на Рефале (пример 2), решающет ту же задачу (цифры справа обозначают номера строк и к программе не относятся).

Программа на Рефале заметно меньше. Разберемся в ней.

Строки 1 и 6 - стандартный заголовок и конец программы (start и end - ключевые слова, reverse - название программы). Строчные и заглавные буквы в Рефале равноправны, поэтому Reverse и reverse - одно и то же имя. Сама программа содержит функции, состоящие в свою очередь из правил замены. Имена функций должны начинаться с первой позиции строки. В программе reverse есть две функции - init и rever. В функции init (строка 3) одно правило, в функции rever два правила (строки 4 и 5).

Правило в программе имеет вид:

шаблон_символьной_строки = чем_заменяется

И левая его часть (шаблон_символьной_строки), и правая могут быть пустыми - например, в строке 3 левая часть, в строке 5 обе. Правило не должно начинаться с первой позиции строки. Дальнейший разбор программы

невозможен без рассказа о том, с чем работает любая программа на Рефале.

В отличие от процедурных ЯП, в которых есть типы, переменные, константы и т.д., в Рефале есть единственный вид данных (к тому же в единственном экземпляре), так называемое, "поле зрения". Это обыкновенная

```
reverse     start                   1
            entry   Init                2
Init        = {rever 'abc'}             3
rever       s0 e1 = {rever e1} s0       4
            =                           5
            end     reverse             6
```

▲ **Пример 2. Программа FIRST**

символьная строка, то есть последовательность символов. Поле зрения состоит из символов нескольких типов. Для понимания программы нам понадобится пока один тип - символы-литеры. Это просто символы, например, 'a', 'A', '1', ' ' (пробел), '''' - они пишутся в апострофах (последний символ обозначает одиночный апостроф - как в Паскале). Между несколькими подряд идущими символами апострофы можно опускать, то есть 'a' 'b' 'c' и 'abc' - одни и те же строки, но вторая значительно короче. При этом между символами в апострофах должен быть по меньшей мере один пробел, иначе два подряд идущих апострофа воспринимаются как один. Таким образом, строка 'a''b''c' означает a'b'c, а не abc.

Но для работы программы одной строки в поле зрения совершенно недостаточно - необходима еще одна конструкция. Условно можно назвать ее "вызов функции с параметрами". В сущности это не вызов функции в классическом смысле, а просто указание о том, какое множество правил будет обрабатывать эту строку. Вызов функции "обрамляет" строку в поле зрения и выглядит так:

{ имя_функции символьная_строка }

Например, в вышеописанной программе FIRST - это строка 3:

{ rever 'abc' }

Можно преобразовать только ту строку, которая стоит в фигурных скобках. Символы, не обрамленные такими скобками, остаются в поле зрения до конца работы программы без изменений, до них вообще уже никак нельзя добраться. Эта конструкция - вызов функции - может быть сколь угодно вложенной.

Теперь более подробно о строении правила. Шаблон (левая часть правила) - это последовательность специальных обозначений - констант и переменных. Контекст, о котором говорилось выше, и шаблон - это одно и то же. Просто, когда речь шла о некоторых основных идеях Рефала, уместно было употреблять термин "контекст". Далее же речь идет об одном из возможных способов реализации контекста - о "шаблоне".

Константы являются теми же символами, что и символы из поля зрения, и обозначаются так же. Они сужают возможный диапазон значений для обрабатываемой строки, служат "каркасом" шаблона, придают ему структуру. Например, запись какой-либо константы первым символом правила означает требование, чтобы строка-параметр начиналась именно с этого символа. Рассказывать о константах отдельно нет смысла, поэтому сразу переходим к переменным.

Переменные в правиле фигурируют в качестве символов и целых групп символов поля зрения. Они являются как бы "наполнением" структуры шаблона (составленной из констант) и обозначают сразу множества допустимых символов, причем иногда бесконечные. Таким образом, контекст представляется в виде структуры символов, состоящей из двух частей - постоянной и переменной. Имена переменных состоят из двух символов - первый обозначает тип, второй собственно имя. Например, в строке 4 программы FIRST есть две переменные - s0 и e1. Переменные локальны в правиле, то есть если бы в функции rever было еще одно правило, то в нем можно было бы использовать эти имена. В качестве имен переменных употребляются буквы латинского алфавита и цифры. Это значит, что в одном правиле не может быть более 36 переменных. (Этого кажется мало, но вообще-то хватает).

Символы "s" и "e" в именах этих переменных обозначают типы (не путайте с типами символов в поле зрения). Тип s - это один символ. Тип e - несколько символов, а именно, ноль или более. Таким образом, в программе FIRST строка 4 (в левой части правила) обозначает, что строка-параметр, переданная функции rever, поделится этим правилом на две части - первый символ s0 и остальные символы e1. Эта строка заменяется правой частью правила на вызов функции rever от "остатка" строки - переменной e1, а первый символ s0 будет стоять последним. Этот символ, не попадая ни в какие фигурные скобки (еще называемые функциональными), остается в поле зрения без изменений до конца работы программы.

Алгоритм, реализованный этой программой, оказывается тривиальным: взять первый символ и переставить его с остатком, далее применить это действие к остатку и т.д. На каждом шаге длина обрабатываемой строки уменьшается - значит алгоритм когда-либо завершится. Однако одного этого правила не хватает. Что случится, если строка-параметр в функции rever будет отсутствовать, то есть будет иметь нулевую длину? А при работе с таким алгоритмом, постоянно уменьшая строку, это обязательно произойдет. В этом случае первое правило функции rever (строка 4) становится неприменимым, так как переменная s0 не может получить

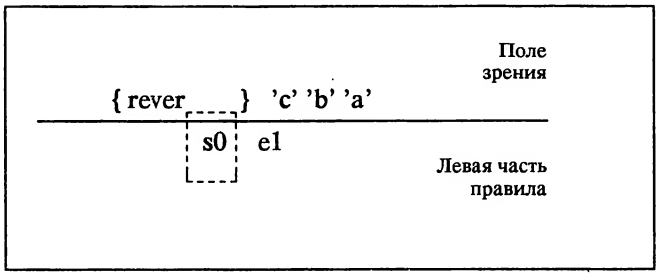


Рис.1. Строка–параметр функции rever отсутствует;переменная s0 не отождествляется

значения - строка отсутствует и неоткуда взять символ (рис. 1).

Но в этой функции есть еще второе правило (строка 5), применимое как раз к этой ситуации - в нем левая часть пуста (рис. 2).

Вообще поиск применимого правила происходит последовательно. Сначала проверяется первое правило. Если оно неприменимо, то Рефал пытается применить второе и т.д.

Теперь можно рассказать и о том, как работает программа. Она работает "по тактам". Сначала в поле зрения ищется самая внутренняя левая пара скобок { }. Потом происходит вызов функции из этих скобок и поиск нужного (применимого правила). В случае если среди правил этой функции есть применимое, то данная пара скобок со всем содержимым заменяется согласно правой части этого правила. Такт заканчивается, и далее все повторяется сначала. Если же применимого правила нет, то программа завершается. Если в поле зрения нет функциональных скобок, то программа тоже завершается. В обоих случаях на экран выдается содержимое поля зрения. Внимательный читатель уже, наверное, заметил, что в описании работы не хватает существенного момента - начальной инициализации, установки поля зрения. Она выглядит так: в поле зрения помещается вызов той функции (без аргумента!), чье имя стоит после ключевого слова "entry". Например, перед началом работы нашей программы поле зрения имеет вид:

{ init }

Проследим работу нашей программы после инициализации. В функции init должно быть правило с пустой левой частью, как и есть в нашем случае (строка 3). Символы 'abc' в строке 3 программы - константы. (Константы могут быть не только в правой, но и в левой части правила.) После выполнения этого правила поле зрения такое:

{ rever 'abc' }

На этом такте произойдет обращение к функции rever с параметром 'abc' (рис. 3).

В функции rever применимо первое правило - переменная s0 обозначает 'a', как говорят, отождествится с 'a', переменная e1 отождествится с 'bc'. Поле зрения после выполнения этого правила:

{ rever 'bc' } 'a'

Опять выполняется первое правило функции rever. Переменная s0 получает значение 'b', e1 - 'c'. Символ 'a' остается на месте, его из функциональных скобок вообще не видно. Поле зрения становится таким:

{ rever 'c' } 'b' 'a'

Происходит то же самое, s0 обозначает 'c', e1 пустую строку. Опять переменная s0 "выносится" за функциональные скобки, но перед символами 'b' 'a':

{ rever } 'c' 'b' 'a'

Вызывается функция rever. Но на этот раз первое правило неприменимо, так как переменная s0 не может получить значение. Это правило отвергается, а попытка применить следующее правило (строка 5) оказывается успешной, и вся конструкция { rever } заменяется на пустую строку.

В поле зрения остается

'c' 'b' 'a'

На этом такте сначала опять происходит поиск функциональных скобок { }. Не найдя их, Рефал прекращает выполнение программы и выдает на экран результат (удалив несущественные апострофы):

'cba'

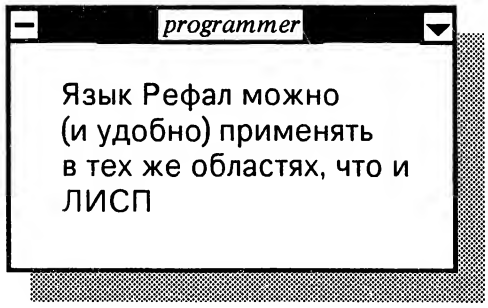
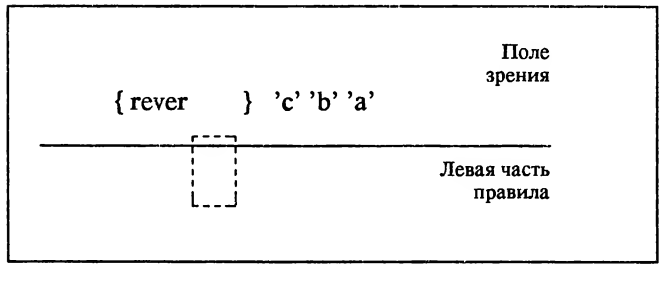


Рис.2.Пустая строка–параметр отождествляется с пустой левой частью правила



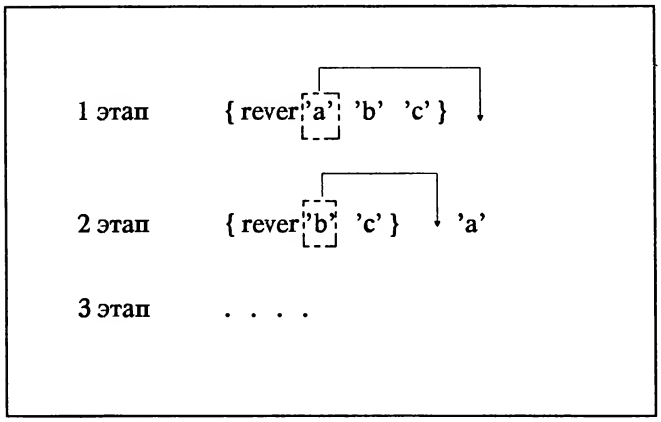


Рис.3. Работа функции rever

Если Вы дочитали до этого места, то, возможно, Вам захотелось написать на Рефале что-нибудь самим. Я предлагаю следующее упражнение: программа получает на входе строку из двух символов, скажем, 'a' и 'b', и переставляет символы 'a' в начало, а 'b' в конец строки, выполняя, таким образом, что-то вроде сортировки. Одно из возможных решений этой задачи приведено ниже (пример 3).

На каждом такте в поле зрения находится только одна пара функциональных скобок (т.е. одна открывающая и одна закрывающая скобки). Слева от нее располагаются буквы 'a', справа - буквы 'b', это обеспечивает первые два правила функции sort. Эти правила работают, так сказать, симметрично. Первое правило применимо, если первым символом строки является 'a'. В этом случае он выносится перед функциональными скобками. Если первый символ не 'a', то второе правило поставит 'b' после них, ведь строка состоит только из 'a' и 'b'. В обоих случаях первый символ исключается из обрабатываемых, строка уменьшается. И, наконец, уже знакомое нам третье правило - обычное завершающее правило в таких программах. Запустив эту программу, мы увидим ожидаемый ответ: 'aaabb'.

Но эти две программы продемонстрировали лишь некоторые черты Рефала. Рассмотрим теперь его вычислительные возможности и для этого напишем еще одну программу. Она получает на входе символьную строку, представляющую номер, и определяет, является ли он "счастливым". (Счастливый номер - это тот, в котором сумма цифр, стоящих на четных местах, равна сумме цифр, стоящих на нечетных местах).

Кроме символов-литер нам понадобится новый тип - символы-числа. Они обозначаются так: /0/, /1/, /1991/. Символы - числа могут быть только целыми неотрицательными. Употребление символов-чисел ничем не отличается от употребления обыкновенных символов, они точно так же используются в правилах, в частности, переменные типа "s" и "e" могут отождествиться с числами. Отрицательные числа представляются комбинацией двух символов, символа-литеры '-' и соответствующего числа. Вот только отождествить отрицательное число с одной переменной типа "s" не удастся, приходится делать это с помощью двух переменных символьного типа. Вообще крайне неудобно, что положительные числа представлены одним символом, а отрицательные - двумя. Несколько улучшает ситуацию тот факт, что положительные числа можно тоже представить двумя символами - символом литерой '+' и самим числом, аналогично отрицательным. Разумеется, нельзя путать числа и литеры: /123/ и '123' означают совершенно разные объекты, даже по количеству элементов. Число /123/ - это один символ, а '123' - три символа.

Символы в поле зрения (и соответственно в правилах) можно группировать, используя специальные символы - круглые скобки ( ). Их можно использовать вперемежку с функциональными скобками, только нужно помнить о том, что к каждой открывающей скобке должна быть такая же закрывающая. Таким образом, поле зрения и правила могут представлять собой весьма сложные иерархические структуры. Заключать эти скобки в апострофы нельзя, так как в таком случае это будут уже самые обыкновенные символы-литеры '(' и ')', которые не структурируют поле зрения.

В Рефале есть целый набор встроенных функций. Их вызов выглядит так же, как и вызов функций, описанных в программе.

Среди них есть и арифметические - сложение (add), вычитание (sub), умножение (mul) и деление нацело (div). Вызываются они одинаково:

```
{ add ( число_1 ) число_2 }
```

Это обращение к функции сложит число_1 и число_2 и поместит результат в поле зрения на место фигурных скобок. Аргументы должны быть символами-числами,

```
abSort          start

        entry   Input

Input   ={sort 'babaa'}

sort
        'a' e0 = 'a' {sort e0}
        'b' e0 = {sort e0} 'b'
        =
```

Пример 3. Программа SORT

причем они могут быть со знаком не только '-', но и '+'. Обратите внимание, что первый аргумент стоит в скобках. Существуют и функции преобразования символьной строки в число и наоборот - это функции cvb (ConVert to Binary, из строки в число) и cvd (ConVert to Decimal, из числа в строку). Обращение к ним тоже одинаково:

```
{ cvb 'строка' }

{ cvd знак /число/ }
```

Аргумент первой функции может быть со знаком '-' или '+', а может быть и без знака, в этом случае результат будет положительным числом. В вызове второй функции знак также можно опускать.

Итак, следует приступать к написанию программы. Первые несколько строк пишутся без труда, мы их неоднократно встречали:

```
Lucky                start

                entry go

go              = {main '1234563'}
```

Алгоритм, сразу приходящий на ум, такой: отрезать два символа от номера, первый прибавить к счетчику цифр на нечетных позициях, второй - к четному счетчику. Так повторять несколько раз, пока можно отрезать два символа. Оставим пока в стороне завершение этого цикла, им мы займемся дальше. Необходимо завести два счетчика. В Рефале это можно сделать только в поле зрения. Счетчики, очевидно, будут символами-числами, и разместить их будет удобно поблизости от обрабатываемой строки. Пусть они находятся перед ней, а чтобы отделить их от нее, заключим счетчики в круглые скобки. Заодно решим, что счетчики - это только положительные числа, поэтому символ '+' перед ними не нужен, и мы его опустим.

Тогда понадобится еще одна инициализация для счетчиков. Сделаем ее в функции main, а сама функция будет состоять из одного правила:

```
main e0 = {loop  (/0/ /0/)  e0 }
```

Это правило применимо всегда, поскольку переменная типа "e" может быть любой длины. После выполнения этого правила в поле зрения появятся счетчики, установленные в ноль. Функция loop будет основной в этой программе.

Напишем основное правило функции loop (то, которое отрезает пару первых символов от номера). Многоточием обозначены неясные пока места (рис. 4, вариант 'а'):

```
loop            ( s0 s1 ) sa sb e2 = {loop ( ... ) e2 }
```

Понятно, что вместо многоточия следует поставить новые значения счетчиков s0 и s1. Нужно будет использовать функцию cvb, чтобы преобразовать переменную sa (обозначающую символ) в число, а потом сложить результат с s0:

```
{ add ({cvb sa}) s0 }
```

Далее поместить результат этого вызова в качестве нового значения s0. Переменные sb и s1 обрабатываются аналогично. Таким образом, все правило будет очень длинным. Чтобы длинные правила хорошо читались, их можно переносить на несколько строк, вставив знак + в любом удобном месте, кроме, конечно, имен функций,

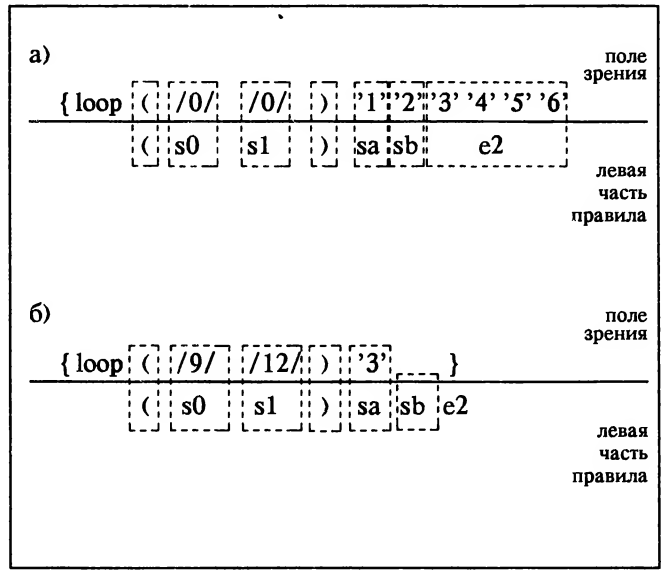


Рис.4. Процесс отождествления:
а) удачно;
б) неудачно (sb не получает значение)

которые нельзя переносить, имен переменных и строковых констант в апострофах. Знак +, находящийся последним в строке, означает, что правило продлевается на следующую строку. Если он не будет последним, то Рефал выдаст сообщение об ошибке. Обратите внимание, что этот знак не заключен в апострофы. С учетом вышесказанного все правило можно написать так:

```
loop    ( s0 s1 ) sa sb e2 = {loop +
          ( {add ({cvb sa}) s0} +
          { add ({cvb sb}) s1} ) e2}
```

Правило получилось громоздким и плохо читаемым, но все программы на Рефале, работающие с числами, несколько неуклюжи.

В некоторых случаях это правило неприменимо, а именно в двух: если в номере осталась одна цифра (если с самого начала в номере было нечетное число цифр, рис. 4 вариант 'б') или ни одной. Сделаем правило для первого случая. Оно выглядит также, как и первое правило функции loop, только увеличить надо будет один счетчик (для цифр на нечетных местах):

```
( s0 s1 ) sa = { loop ({add ({cvb sa}) s0} s1) }
```

Осталось написать завершающие правила. Когда строка кончится, то нужно проверить содержимое счетчиков, если они равны, то номер счастливый:

```
( s0 s0 ) = 'yes'
```

Это не опечатка. Строка обозначает, что два символа из поля зрения должны совпадать (рис. 5).

Несколько вхождений одной и той же переменной в левую часть правила должны обозначать одинаковые объекты, в данном случае два одинаковых числа. Для символов-чисел это означает, что они равны. В Рефале есть, конечно, и встроенные функции сравнения чисел, но этот способ изящнее. Если содержимое счетчиков разное, то номер не счастливый:

```
( s0 s1 ) = 'no'
```

Переменные s0 и s1 могут принимать всевозможные значения, в том числе и одинаковые. Поэтому из двух завершающих правил это правило должно стоять последним - иначе срабатывать будет только оно.

Приведем программу полностью (пример 4).

Вот мы и познакомились с Рефалом. Знакомство было, конечно, неполным, да и невозможно в короткой

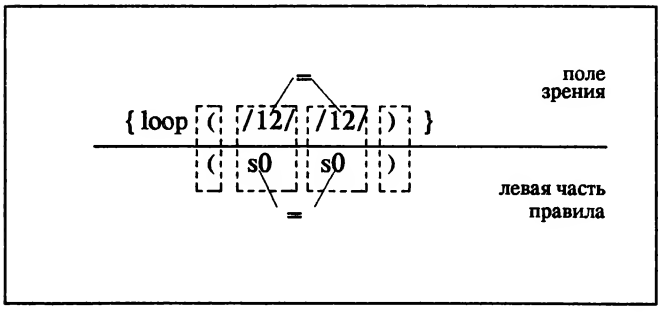


Рис.5. Процесс отождествления при совпадении переменных в поле зрения

статье сколько-нибудь подробно описать любой достаточно большой язык программирования. Кроме того, многие части Рефала были опущены. Например, когда шла речь о поле зрения, о том, что это единственное данное, имеющееся в распоряжении, то это было не совсем верно, так как в Рефале есть возможность завести переменные, сходные с переменными в обычном, процедурном языке программирования. Для этого существует дополнительная память. Кроме этого, переменные в наших правилах были самого общего вида. Их можно специфицировать, указывая, какие значения они могут принимать, какие не могут.

Хотя на Рефале удобно писать обработку текстов, автору неизвестно, существуют ли какие-либо программы, ориентированные хоть на какое-то широкое применение. Сфера приложения Рефала ограничивается сугубо научными разработками. Рефал используется в искусственном интеллекте при работе с формальными грамматиками и создании компиляторов и интерпретаторов.

```
Lucky       start

      entry  go

go          = {main '1234563'}

main        e0 = {loop (/0/ /0/) e0 }

loop        ( s0 s1 ) sa sb e2 = {loop +
            ( {add ({cvb sa}) s0} +
            {add ({cvb sb}) s1} ) e2}
            ( s0 s1 ) sa = {loop ({add ({cvb sa}) s0} s1) }
            ( s0 s0 ) = 'yes'
            ( s0 s1 ) = 'no'

            end   Lucky
```

Пример 4. Программа Lucky

Рефал можно (и удобно) применять практически в тех же областях, что и ЛИСП. Тем не менее Рефал не является широко распространенным языком, главным образом вследствие того, что он был создан у нас в стране, где, к сожалению, силен бюрократический подход к науке. ЛИСП же, возникнув в Америке, снискал себе значительно большую популярность.

История языков программирования
Хронологическая таблица

1804 год

• *Французский изобретатель Жозеф Мари Жаккар создал "программно-управляемый" ткацкий станок. Для управления станком использовались перфокарты, соединенные друг с другом в виде ленты. Деревянные шпильки "читающего устройства" станка по расположению отверстий в перфокарте определяли, какие нити следует поднять, а какие опустить для получения нужного узора.*

1890 год

• *Созданная американским изобретателем Германом Холлеритом электромеханическая счетная машина - табулятор, управляемая перфокартами, была использована для составления таблиц с результатами переписи населения США. Основанная Холлеритом фирма по производству табуляторов впоследствии превратилась в корпорацию International Business Machines (IBM).*

1931 год

• *Американский математик и логик Алонзо Черч описывает лямбда-исчисление, оперирующее тремя элементами : символами, скобками и обозначениями функций, выражаемыми греческой буквой "лямбда". Лямбда-исчисление легло в основу системы обозначений языка Лисп.*

1936 год

• *Двадцатипятилетний студент Кембриджского университета англичанин Алан Тьюринг опубликовал статью "О вычислимых числах", в которой рассматривалось гипотетическое устройство ("машина Тьюринга"), пригодное для решения любой разрешимой математической или логической задачи, - прообраз программируемого компьютера.*

1941 год

• *Немецкий инженер Конрад Цузе построил действующий компьютер Z3, в котором использовалась двоичная система счисления. Программы записывались на перфоленте, изготовленной из бракованной кинопленки.*

1945 год

• *В Высшем техническом училище Пенсильванского университета (США) физик Джон Мочли и инженер Проспер Экерт построили полностью электронную машину "Эниак". Для задания программы было необходимо вручную установить тысячи переключателей и воткнуть сотни штекеров в гнезда контактной панели.*

• *1 июня 1945 года. Разослан отчет американского математика Джона фон Неймана "Предварительный отчет о машине Эдвак", содержащий концепцию хранения команд компьютера в его собственной внутренней памяти.*

1948 год

• *21 июня 1948 года. В Манчестерском университете (Великобритания) на машине "Марк-1" выполнена первая в мире хранимая в памяти машины программа - поиск наибольшего сомножителя заданного числа.*

1949 год

• *Под руководством Мориса Уилкса создан компьютер "Эдсак". Проектировщики "Эдсака" ввели систему мнемонических обозначений, где каждая машинная команда представлялась одной заглавной буквой, и автоматизировали настройку подпрограмм на определенное место в памяти. Морис Уилкс назвал мнемоническую схему и библиотеку подпрограмм собирающей системой (assembly system) - отсюда слово "ассемблер".*

• *В Филадельфии (США) под руководством Джона Мочли создан "Краткий код" - первый примитивный интерпретатор языка программирования.*

1951 год

• *В фирме Remington Rand американская программистка Грейс Хоппер разработала первую транслирующую программу. Хоппер назвала ее компилятором (compiler - компоновщик).*

1956 год

• *Группа Грейс Хоппер создает компилятор FLOW-MATIC для языка программирования, использующего обычные английские слова.*

1957 год

• *На 20-м этаже штаб-квартиры фирмы IBM на Мэдисон-авеню в Нью-Йорке родился язык Фортран (FORmula TRANslation - трансляция формул). Группой разработчиков руководил 30-летний математик Джон Бэкус. Фортран - первый из "настоящих" языков высокого уровня - остается популярным и сегодня. "В известном смысле все исследования в области программирования с 1957 года были вызваны стремлением избавиться от недостатков Фортрана" (Лоуренс Г. Теслер).*

1960 год

• *Январь 1960 года. Тринадцать европейских и американских специалистов в Париже утвердили стандарт языка Алгол-60 (ALGOrithmic Language - алгоритмический язык). При описании синтаксиса языка была использована система обозначений, разработанная американцем Джоном Бэкусом и уточненная датским астрономом Питером Науром. Форма Бэкуса-Наура или бэкусова нормальная форма (БНФ) стала стандартом при записи синтаксиса языков программирования. Алгол оказал сильнейшее воздействие на последующие языки программирования. Среди фундаментальных идей, введенных в Алголе - блочная структура программ и рекурсия.*

• *Апрель 1960 года. Опубликованы спецификации языка Кобол (COmmon Business Oriented Language - универсальный язык для бизнеса). Язык был разработан комитетом CODASYL, созданным Министерством обороны США совместно с крупнейшими фирмами - производителями компьютеров.*

• *Опубликована статья американского математика Джона Маккарти "Рекурсивные функции и символьные вычисления", в которой излагалось математическое обоснование языка программирования Лисп (LISt Processing - обработка списков). Основные составляющие Лиспа - списки, рекурсивные функции и система обозначений, заимствованная из лямбда-исчисления. Сегодня Лисп - основной язык программирования в исследованиях по искусственному интеллекту.*

1964 год

• *Февраль 1964 года. Группа специалистов под руководством Джорджа Рэдина завершила работу над спецификациями языка ПЛ/1 (Programming Language One) - универсальным языком программирования для компьютеров "Системы 360" фирмы IBM. Язык претендовал на сочетание лучших черт Алгола, Фортрана и Кобола, но оказался чрезмерно переусложнен.*

• *1 мая 1964 года. Профессора Дартмутского колледжа (США) Томас Курц и Джон Кемени завершили работу над первой версией языка Бэйсик (BASIC - Beginners All-Purpose Symbolic Code).*

1965 год

• *Американец Дж. Робинсон публикует статью "Машинно-ориентированная логика, основанная на принципе*

резолюции", содержащую математические основы логического программирования.

1968 год

• Март 1968 года. Опубликована историческая статья голландского программиста Эдсгера Дейкстры "Go to statement considered harmful" ("О вреде оператора GO TO").

1969 год

• Фирма IBM выпустила для широкого пользования язык АПЛ (A Programming Language), разработанный несколькими годами ранее Кеннетом Айверсоном. Нестандартный набор символов и удивительная компактность записи привели к тому, что АПЛ по внешнему виду резко отличается от других языков программирования.

• Уроженец Южной Африки профессор математики и педагогики Массачусетского Технологического института Сеймур Пейперт создал на основе Лиспа язык Лого ("лого" по-гречески означает "слово"). Лого - язык управления "черепашкой" - треугольным значком, который может двигаться по экрану компьютера. Язык настолько прост, что создавать на нем программы может даже маленький ребенок.

1970 год

• Чарльз Мур, сотрудник Национальной радиоастрономической обсерватории в Аризоне, создает язык Форт (FORTH - вперед). Он развивал язык несколько лет как способ повышения собственной производительности. Первые программы на Форте управляли в реальном времени системой наведения 11-метрового телескопа обсерватории Китт-Пик. Характерные черты Форта - простота, гибкость и возможность дополнения языка любыми средствами по мере необходимости. Основная структура данных в Форте - стек.

• Один программист - Т.Бейкер из фирмы IBM под руководством Х.Милса за 22 месяца создал систему автоматизации хранения справочного материала (газетных вырезок) для газеты "Нью-Йорк таймс" (83 тысячи операторов на языке высокого уровня) - триумф методов структурного программирования.

1972 год

• Опубликована работа Эдсгера Дейкстры "Заметки по структурному программированию", содержащая блестящее описание основных идей структурного программирования.

• Француз Алан Колмероэ из университета в Лумини (Марсель) разработал язык Пролог (PROgramming in LOGic - программирование в логике) - первый язык логического программирования. Пролог - непроцедурный язык; в программе на Прологе описывается желаемый результат, но не задается способ его получения.

• 31-летний специалист по системному программированию из фирмы Bell Labs Деннис Ритчи разработал язык программирования Си.

1973 год

• Швейцарский специалист по программированию Никлаус Вирт опубликовал "Пересмотренное сообщение", определившее точный стандарт языка Паскаль. Строгий стиль Паскаля был с восторгом принят приверженцами структурного программирования.

1976 год

• Опубликована книга Эдсгера Дейкстры "Дисциплина программирования", посвященная фундаментальным вопросам построения программ и представляющая принципиально новую точку зрения на существо программирования.

1978 год

• Выходит в свет статья британского программиста и математика Чарльза Энтони Ричарда Хоара "Взаимодействующие последовательные процессы", заложившая математические основы программирования для параллельных вычислительных систем.

1979 год

• 2 мая 1979 года. Объявлен победитель конкурса по созданию языка для программирования вычислительных систем Министерства обороны США. Им стал язык Ада, разработанный фирмой CII Honeywell-Bull под руководством француза Жана Ишбиа. Основные конструкции языка заимствованы из Паскаля, но в целом язык оказался чрезвычайно большим и сложным, что вызвало критику автора Паскаля Никлауса Вирта. Распространенность Ады за пределами военных приложений (системы связи и вооружений) невелика.

1981 год

• Опубликован учебник профессора Корнеллского университета (США) Дэвида Гриса "Наука программирования"."Если считать переломную книгу Э.Дейкстры интеллектуальным откровением, то книга Д.Гриса - это апостольское деяние, направленное на то, чтобы превратить учение одиночки в мировую религию" (А.П.Ершов).

• Март 1981 года. Передан пользователям компилятор с языка программирования Модула-2, разработанного Никлаусом Виртом и являющегося наследником языков Паскаль и Модула. Основные особенности Модулы-2 - реализация программ в виде набора самостоятельных модулей и мультипрограммирование.

1983 год

• Выпущен на рынок и приобрел широкую известность язык Смолток (Smalltalk), разрабатывавшийся с начала 70-х годов Аланом Кеем и группой его коллег из Исследовательского центра фирмы Xerox в Пало-Альто (США), - первый объектно-ориентированный язык программирования.

1984 год

• Бывший студент Никлауса Вирта французский математик и саксофонист Филип Кан основывает фирму Borland International и начинает продажу по почте компилятора Турбо-Паскаль для персональных компьютеров. Компактный, быстродействующий и удивительно дешевый Турбо-Паскаль только за первый месяц принес Кану заказов на 150 тысяч долларов. Местные банки даже заподозрили его в почтовом мошенничестве и поначалу отказывались оплачивать чеки и кредитные карточки. Турбо-Паскаль вывел Borland в ряды основных производителей программного обеспечения.

• Фирма British semiconductor manufacturer INMOS разрабатывает язык Оккам - "язык ассемблера" для вычислительных систем, построенных из множества параллельно работающих специальных микропроцессоров - транспьютеров. Язык назван в честь средневекового английского философа-схоласта и логика Уильяма Оккама (1285 - 1349) и основан на математической теории Ч.А.Р. Хоара. Основное понятие языка - процесс. Процессы могут выполняться как последовательно, так и параллельно и взаимодействуют с помощью каналов.

1985 год

• Франко-американская научная экспедиция, состоявшая из исследовательского судна ВМС США "Кнорр" и подводного аппарата "Арго" обнаружила на дне Атлантического океана затонувший в 1912 году "Титаник". Это событие явилось выдающимся достижением компьютерной технологии. Компьютер подводного аппарата был запрограммирован с помощью языка Форт, компьютер на борту судна "Кнорр" - с помощью языка Си, телеметрическая система передачи сигналов по кабелю между судами - с помощью языка ассемблера.

И. Гун

МЫ ДРУЖИМ С ГРАФИКОЙ!

или введение в использование

Стоит написать с десяток программ, как с удивлением обнаруживаешь, что компьютер "решает" поставленные задачи и тут же сам "ставит" перед тобой новые. Конечно, компьютер еще не способен мыслить как человек, но и человек не всегда способен представить себе глубину затрагиваемой проблемы.

Пусть, к примеру, необходимо проанализировать поведение сложной строительной конструкции при различных внешних воздействиях на нее. Прежде чем компьютер "выдаст" первые результаты, программист-математик проделает сложную, многоэтапную работу: выведет уравнения; подберет метод решения; алгоритмизирует задачу; закодирует на специальном языке; отладит программу ... Но вот получены результаты (обычно в виде цифр). Как их интерпретировать? Способен ли это сделать кто-либо еще кроме автора программы?

К сожалению для программиста ценность проделанной работы зависит не только от того, насколько точны результаты расчета, сколько от того, как легко эти результаты могут быть восприняты пользователем или даже самим разработчиком.

Проблема представления результатов - это та плоскость, на которую проектируется большинство расчетных задач.

В чем состоит ее решение? Из полученных цифр (результатов расчета) необходимо сконструировать графический объект. Наш мозг сам проведет его анализ и, если объект удачно построен, мы сможем принять нужные и верные решения, не обращаясь к кому-либо за помощью.

На каких этапах программирования и в каких задачах возникает потребность в графической визуализации?

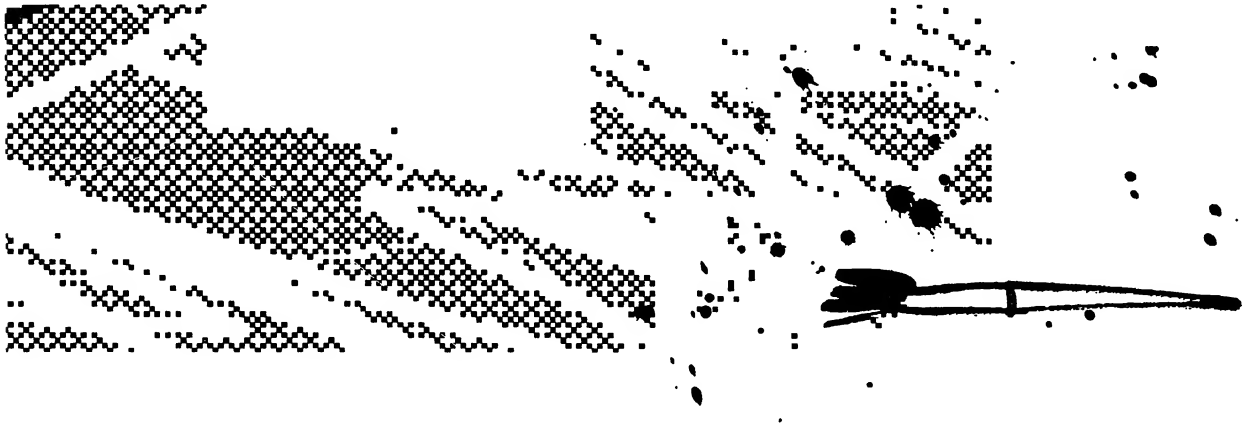
Обращаясь к собственному опыту программирования, я бы посоветовал: ИСПОЛЬЗУЙТЕ ГРАФИКУ В КОНЕЧНОМ ПРОДУКТЕ И ПРИ ОТЛАДКЕ ПРОГРАММ И ПРИ ИХ ТЕСТИРОВАНИИ. Даже если Ваши программы не расчетного плана, но достаточно велики (более 10 тыс. оп.), используя графические возможности PC, Вы сократите время разработки.

Допустим, Вы приняли мой тезис, но не знаете как его реализовать на персональном компьютере.

Использование графики требует специальных пакетов, включающих помимо всего прочего (трансляторы, базы данных и пр.) библиотеки графических примитивов. Они реализованы почти во всех трансляторах с языков программирования: Quic Basic и Pascal, Turbo Pascal, Microsoft Fortran и т.д.. Но даже при использовании этих прекрасных пакетов, у Вас будет уходить уйма времени на программирование, если Вы будете приспосабливать графические примитивы каждый раз к новой задаче.

Нисколько не претендуя на полноту (теме компьютерной графики посвящены тома литературы), мы остановимся на основных моментах конструирования графического пакета. Скопировав примеры приведенных программ, Вы получите простую, но полезную графическую утилиту, которую сами сможете развить в нужном для себя направлении.

библиотек графики Turbo Pascal



Turbo Pascal 5.5

Выбор Turbo Pascal фирмы Borland не случаен. Это, пожалуй, самая удачная версия транслятора, содержащая интегрированную среду и разнообразнейшую документацию по ее использованию и функциям библиотек.

Чтобы "запустить" программу в этой среде, ее порой не обязательно вводить с клавиатуры.

Скопируем, к примеру, программу черчения линии:

- нажмем клавишу F10;
- в режиме File выберем опцию New;
- нажмем одновременно клавиши <Ctrl> и F1;
- выберем procedure;
- найдем в списке LineTo;
- выберем Exampl и с помощью клавиш управления курсором и клавиш C,B перенесем пример в окно.

На экране должен появиться листинг *:

```
uses Graph;

var
  Gd, Gm : Integer;
begin
  Gd := Detect;
  InitGraph(Gd, Gm, '');
  if GraphResult <> grOk then Halt(1);
  MoveTo(0,0); { Upper-left corner of port }
  LineTo(GetMaxX, GetMaxY);
  ReadLn;
  CloseGraph;
end.
```

* Во всех случаях, когда рекомендуемые действия не приводят к успеху, необходимо проверить правильность установленной конфигурации, нажав F10, O, D.

Нажав F9, <Ctrl>-F9, на терминале прочертится диагональная линия.

Она появилась на экране в результате работы двух процедур - MoveTo и LineTo. Первая установила координаты начальной точки линии, а вторая, получив координаты второй точки линии, и прочертила линию.

Без преувеличения можно сказать, что LineTo - это основная процедура (все изображения можно построить из отрезков), однако, для того чтобы использовать ее более полно, необходимо проделать некоторую дополнительную работу.

Мировые координаты и окна

Графический терминал способен отобразить только те обьекты, координаты которых укладываются в узкий интервал целых чисел (для адаптера EGA границы интервала по горизонтали 0..650, по вертикали 0..350). Конечно это существенное ограничение и, чтобы преодолеть его, приходится перед вычерчиванием объекта преобразовывать его координаты.

Возможные численные значения координат объекта называют МИРОВЫМИ КООРДИНАТАМИ.

Если, к примеру, нас интересует полученная предприятием прибыль за первое полугодие, то в качестве геометрического объекта, отражающего динамику деятельности предприятия, можно взять график доходов на каждое первое число каждого месяца.

Все МИРОВЫЕ КООРДИНАТЫ мы никогда не сможем отобразить в ограниченное координатное пространство терминала. Поэтому самый первый шаг построения картинки на терминале - это вырезание ОКНА или наиболее интересующей нас области из МИРОВОГО КООРДИНАТНОГО пространства.

Это сделать совсем просто, если создать структуру, содержащую информацию о границах окна:

```
PortArray : array [1..NumPort] of record
   Port_max_x, Port_min_x,
   Port_max_y, Port_min_y : real
end;
```

Заполнить ее конкретными значениями можно так:

```
with PortArray[2] do begin
   Port_max_x:=1;
   Port_min_x:=6;
   Port_max_y:=10000;
   Port_min_y:=0
end;
```

Если наше гипотетическое предприятие работает из рук вон плохо (с отрицательной нормой прибыли) или наоборот очень хорошо (прибыль более 10000), мы скорее всего не сможем разобраться в этом по изображенному на терминале графику. Ведь все, что не попадает в

```
Procedure MaxMin(var MaxX, MinX, MaxY, MinY : real);
var
  i : word;
begin {MaxMin}
  MaxX:=-1e2;
  MaxY:=-1e2;
  MinX:=1e2;
  MinY:=1e2;
  for i:=1 to NumDot do begin
    if MaxX < InputDateX[i] then MaxX:=InputDateX[i];
    if MaxY < InputDateY[i] then MaxY:=InputDateY[i];
    if MinX > InputDateX[i] then MinX:=InputDateX[i];
    if MinY > InputDateY[i] then MaxY:=InputDateY[i];
  end;
```

Пример 1. Процедура автоматического определения границ окон по входному массиву данных

```
Function NormCoordX(X : real) : real;
begin {NormCoord}
  with PortArray[Port] do
  NormCoordX:=(X-Port_min_x)/(Port_max_x-Port_min_x)
end; {NormCoord}

Function NormCoordY(Y : real) : real;
begin {NormCoord}
  with PortArray[Port] do
  NormCoordY:=(Y-Port_min_y)/(Port_max_y-Port_min_y)
end; {NormCoord}
```

Пример. 2. Преобразование мировых координат в координаты виртуального дисплея

```
Procedure SetViewPort(View_min_x, View_min_y, View_max_x,
        View_max_y : real; Port : word);
begin {SetViewPort}
  with ViewPort[Port] do begin
    MinX:=Round(View_min_x*GetMaxX);
    MinY:=GetMaxY-Round(GetMaxY*View_min_y);
    MaxX:=Round(View_max_x*GetMaxX);
    MaxY:=GetMaxY-Round(GetMaxY*View_max_y);
  end;
end; {SetViewPort}
```

Пример. 3. Задание области вывода на виртуальном дисплее

```
Type
  Adapter =
  (CGA_, EGA_Hi, EGA_Lo, VGA_Lo, VGA_Med, VGA_Hi, Indef);

Procedure SetAdapter (Adapter : TypeAdapter)};
var
  Gd, Gm : Integer;
  DRIVER : procedure;
begin {SetAdapter}
  case Adapter of
    CGA_ : begin
      Gd:=CGA;
      Gm:=CGAC0;
      DRIVER:=CGADRIVER
    end;
    EGA_Hi:begin
      Gd:=EGA;
      Gm:=EGAHi;
      DRIVER:=EGADRIVER
    end;
    EGA_Lo:begin
      Gd:=EGA;
      Gm:=EGALo;
      DRIVER:=EGADRIVER
    end;
    VGA_Lo : begin
      Gd:=VGA;
      Gm:=VGALo;
      DRIVER:=EGADRIVER
    end;
    VGA_Med : begin
      Gd := VGA;
      Gm:= VGAMed;
      DRIVER:=EGADRIVER
    end;
    VGA_Hi : begin
      Gd := VGA;
      Gm:= VGAHi ;
      DRIVER:=EGADRIVER
    end;
    Indef : begin
      Gd := Detect;
      DRIVER:=EGADRIVER
    end;
  end;
  if RegisterBGIDriver(Addr(DRIVER)) < 0 then begin
    Writeln('Ошибка регистрации драйвера...');
    Halt(1)
  end else InitGraph(Gd, Gm, '')
end; {SetAdapter}
```

Пример 4. Программа загрузки драйверов терминальных адаптеров

окно, отсекается, то есть не появляется на экране терминала.

Ситуация, когда заранее трудно предсказать,какие численные значения возникнут в ходе расчета или будут введены с клавиатуры, встречается довольно часто. Как поступать в этом случае?

Пусть входные данные передаются через массив InputDateX и InputDateY:

```
InputDateX, InputDateY : array [1..MaxDot] of real;
```

Процедура MaxMin (пример 1) автоматически заполнит структуру PortArray:

```
with PortArray[NumPort] do
  MaxMin(Port_max_x, Port_min_x, Port_max_y, Port_min_y);
```

Итак, мы умеем вырезать ОКНА из МИРОВОГО КООРДИНАТНОГО пространства и все, что в них попадает, собираемся изобразить на экране дисплея. Но что такое "дисплей" ? Я бы не взялся объяснять новичку его устройство - это достаточно сложно. Однако это не помешает нам использовать его возможности. Во всем мире разработчики программного обеспечения пишут свои программы, используя понятие ВИРТУАЛЬНОГО УСТРОЙСТВА. В нашем случае - это квадратная область, верхний правый угол которой имеет координаты 1.,1., а нижний 0.,0.. Все координаты InputDateX, InputDateY могут быть преобразованы в новые нормированные (пример 2) .

Перед тем как преобразовать нормированные координаты в дисплейные, необходимо условиться, в какой области терминала должен появиться объект. Это легко сделать, используя координаты виртуального дисплея. Нижний и верхний уголы экрана задаются значениями от нуля до единицы, эти значения и передаются процедурой SetViewPort (пример 3). Функции GetMaxX, GetMaxY, используемые здесь, позволяют получить максимальные разрешения по горизонтали и вертикали реального терминального устройства.

Наконец, мы можем записать преобразования нормированных координат в координаты конкретного терминала:

```
Function LineCoordX(Xnorm : real) : integer;
begin {LineCoordX}
  with ViewPort[Port] do
    LineCoordX:=MinX+Round((MaxX-MinX)*Xnorm)
end; {LineCoordX}

Function LineCoordY(Ynorm : real) : integer;
begin {LineCoordY}
  with ViewPort[Port] do
    LineCoordY:=MinY+Round((MaxY-MinY)*Ynorm)
end; {LineCoordY}
```

Если теперь мы заполним массивы входных данных и выполним фрагмент программы :

```
SetViewPort(0,0,1,1,Port);
MoveTo(LineCoordX(NormCoordX(InputDateX[1])),
       LineCoordY(NormCoordY(InputDateY[1])));
for I:=2 to MaxDot do
  LineTo(LineCoordX(NormCoordX(InputDateX[i])),
       LineCoordY(NormCoordY(InputDateY[i])));
```

кривая нарисуется при любых входных данных!

Адаптеры терминалов

До настоящего момента написанный нами код программ не носил ссылок на какое - либо конкретное терминальное устойство. Это правильная практика проектирования программ. Программы, написанные таким образом, легко переносить на компьютеры различных марок. Переписывать приходится лишь небольшие куски кода, использующие драйверы уникальных устройств.

Фирма Borland разработала для своего графического пакета свои драйверы терминальных адаптеров. Файлы с этими программами имеют расширение BGI. Для того чтобы использовать их в своей программе, рекомендуется перетранслировать с помощью программы BINOBJ их в формат обьектного файла и вызывать в программе как внешнюю процедуру.

Проделаем эти действия с драйвером EGA. В командной строке ДОС необходимо ввести:

```
>BINOBJ EGAVGA.BGI EGAVGA.OBJ EGADRIVER
```

Для образования процедуры из внешнего файла достаточно написать:

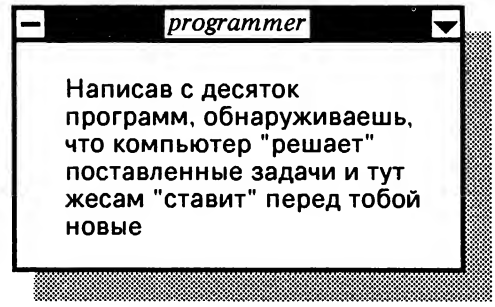
```
Procedure EGADRIVER; external;
{$L EGAVGA.OBJ}
```

Процедура SetAdapter (пример 4) позволяет установить любой из определенных с помощью типа TypeAdapter драйверов, причем графический вывод (пропорции изображения) при этом не изменяется.

Приведенные в конце статьи листинги (пример 5 и листинг в приложении) содержат законченную интерактивную графическую систему. Обладая достаточно скромными возможностями, она содержит тем не менее все подсистемы, присутствующие в серьезных пакетах: меню, ввод данных, вывод двумерных графиков. Построенную по принципу аппаратной независимости, эту программу легко "запустить" и на мониторе с небольшой резрешающей способностью (CGA, MCGA) и на высококачественных терминалах.

Литература

Фоли Дж., вэн Дэм А. Основы интерактивной машинной графики, М.:Мир 1985, 368с., ил.



Пример 5. Программа, иллюстрирующая процедуры графичекой библиотеки SmallGr

```
Program DemoGraph;

uses
  Crt, Graph, SmolGr;

var
  Stop : boolean;

begin
  SetAdapter(EGA_Lo); { установить драйвер терминала }
  Stop:=False;
  repeat
    Port:=4;
    PlayMenuMain; { вывести главное меню выбора }

    case Selector of { выбрать позицию меню }

    Select_Port : begin { выбор окна вывода }
      PlayMenuPort;
      SelectPort(DatePort);
    end;

    Input_Date : begin { ввод данных }
      Port:=3;
      InputDate(DatePort);
    end;

    View_ : begin { визуализация }
      Port:=DatePort;
      PlotGraphic(DatePort);
    end;

    Exit_ : Stop:=True; { выход из системы }

    end;
  until Stop;

  CloseGraph;
end.
```

```
{ Библитотека графики SmallGraph }

Unit SmallGr;
interface
uses Crt,Dos,Graph;
const
  MaxDot = 6; {число точек кривой}
  NumPort = 4; {число окон}
  NumPos = 4;
  Dn = 80; RETURN = 13; Up = 72; {коды клавиш управления курсором}
type
  TypeAdapter = (CGA_, EGA_Hi, EGA_Lo, VGA_Lo, VGA_Med, VGA_Hi, Indef);
  MainSelect = (Select_Port, Input_Date, View_, Exit_);
  Bufer = array [1..MaxDot] of real;
var
  PortArray : array [1..NumPort] of record
    Port_max_x, Port_min_x, Port_max_y, Port_min_y : real
  end;
  ViewPort : array [1..NumPort] of record
    MinX, MinY, MaxX, MaxY : integer
  end;
  InputDateX, InputDateY : Bufer;
  Port, DatePort, Pos : word;
Procedure SetAdapter(Adapter : TypeAdapter); {загрузка драйвера терминала}
Procedure MaxMin(var MaxX, MinX, MaxY, MinY : real); {определение границ окон}
Procedure ClearPort(Port : word); {очистка окна вывода}
Procedure SetViewPort {установление границ вывода}
  (View_min_x, View_min_y,View_max_x, View_max_y : real; Port : word);
Procedure InputDate (PortDate : word); {клавиатурный ввод данных}
Procedure PlotGraphic(Port : word); {рисование графика}
Procedure PlayMenuMain; {изображение основного меню на экран}
Procedure PlayMenuPort; {изображение меню выбора окон}
Function Selector : MainSelect; {выбор позиции основного меню}
Procedure SelectPort(var Port : word); {выбор позиции меню окон}

Implementation

var
  MainMenu : array [1..NumPos] of record
    PenX, PenY : real;
    Text : string
  end;
  PortMenu : array [1..NumPort] of record
    PenX, PenY : real;
    Text : string
  end;
  Procedure EGADRIVER; external;
  {$L EGAVGA.OBJ}
  Procedure CGADRIVER; external;
  {$L CGA.OBJ}
```

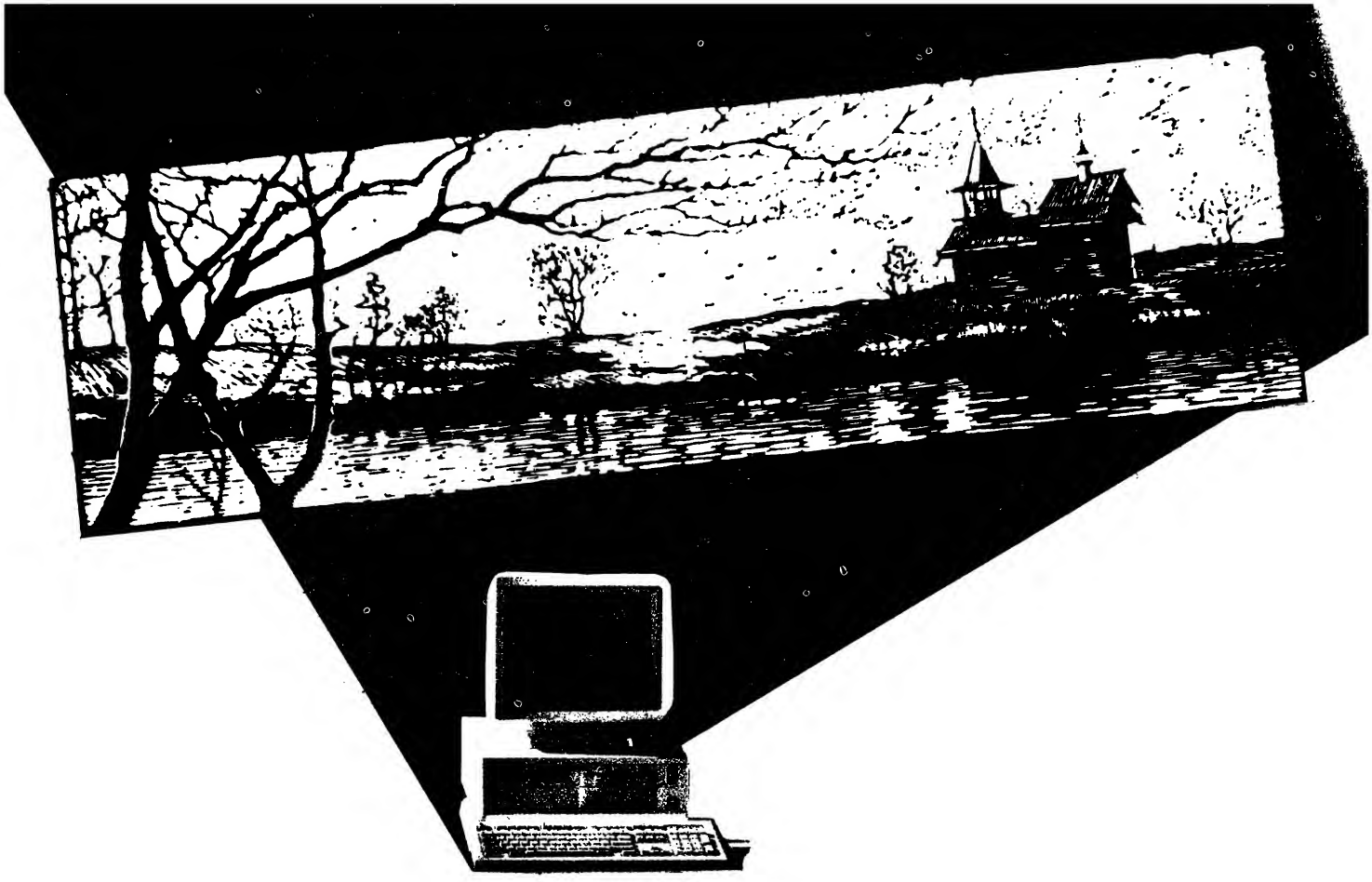
```
Procedure SetAdapter {(Adapter : TypeAdapter)};
var
  Gd, Gm : integer;
  DRIVER : procedure;
begin {SetAdapter}
  case Adapter of
    CGA_ : begin Gd:= CGA; Gm:= CGAC0 ; DRIVER:=CGADRIVER end;
    EGA_Hi : begin Gd := EGA; Gm:= EGAHi; DRIVER:=EGADRIVER end;
    EGA_Lo : begin Gd := EGA; Gm:= EGALo; DRIVER:=EGADRIVER end;
    VGA_Lo : begin Gd := VGA; Gm:= VGALo; DRIVER:=EGADRIVER end;
    VGA_Med : begin Gd := VGA; Gm:= VGAMed; DRIVER:=EGADRIVER end;
    VGA_Hi : begin Gd := VGA; Gm:= VGAHi ; DRIVER:=EGADRIVER end;
    Indef : begin Gd := Detect; DRIVER:=EGADRIVER end;
  end;
  if RegisterBGIDriver(Addr(DRIVER)) <> 0 then begin
    Writeln('Ошибка регистрации драйвера...'); Halt(1)
  end else InitGraph(Gd, Gm, '')
end; {SetAdapter}

Procedure MaxMin(var MaxX, MinX, MaxY, MinY : real);
var
  Dot : word;
begin {MaxMin}
  MaxX:=-1e20; MaxY:=-1e20; MinX:=1e20; MinY:=1e20;
  for Dot:=1 to MaxDot do begin
    if MaxX < InputDateX[Dot] then MaxX:=InputDateX[Dot];
    if MaxY < InputDateY[Dot] then MaxY:=InputDateY[Dot];
    if MinX < InputDateX[Dot] then MinX:=InputDateX[Dot];
    if MinY < InputDateY[Dot] then MinY:=InputDateY[Dot];
  end;
end;{MaxMin}

Function NormCoordX(X : real) : real;
begin {NormCoord}
  with PortArray[Port] do
    NormCoordX:=(X-Port_min_x)/(Port_max_x-Port_min_x)
end; {NormCoord}

Function NormCoordY(Y : real) : real;
begin {NormCoord}
  with PortArray[Port] do
    NormCoordY:=(Y-Port_min_y)/(Port_max_y-Port_min_y)
end; {NormCoord}

Function LineCoordX(Xnorm : real) : integer;
begin {LineCoordX}
  with ViewPort[Port] do
    LineCoordX:=MinX+Round((MaxX-MinX)*Xnorm)
end; {LineCoordX}

Function LineCoordY(Ynorm : real) : integer;
begin {LineCoordY}
```

```
    with ViewPort[Port] do
      LineCoordY:=MinY+Round((MaxY-MinY)*Ynorm)
end; {LineCoordY}

Procedure ClearPort(Port : word);
begin {ClearPort}
  with ViewPort[Port] do begin
    SetFillStyle(SolidFill,Black);
    Bar(MinX,MinY,MaxX,MaxY);
  end;
end; {ClearPort}

Procedure SetViewPort
          (View_min_x, View_min_y, View_max_x, View_max_y : real;Port : word);
begin {SetViewPort}
  with ViewPort[Port] do begin
    MinX:=Round(View_min_x*GetMaxX);
    MinY:=GetMaxY-Round(GetMaxY*View_min_y);
    MaxX:=Round(View_max_x*GetMaxX);
    MaxY:=GetMaxY-Round(GetMaxY*View_max_y);
  end;
end; {SetViewPort}

Procedure MovePen(X,Y : real);
begin {MovePen}
  MoveTo(LineCoordX(NormCoordX(X)), LineCoordY(NormCoordY(Y)));
end; {MovePen}

Procedure MoveText(X,Y : real);
begin {MovePen}
  MoveTo(LineCoordX(X), LineCoordY(Y));
end; {MovePen}

Procedure LinePen(X,Y : real);
begin {LinePen}
  LineTo(LineCoordX(NormCoordX(X)), LineCoordY(NormCoordY(Y)));
end; {LinePen}

Procedure PlotGraphic(Port : word);
var
  Dot : word;
begin {PlotGraphic}
  case Port of
    1 : SetViewPort(0,0.5,0.5,1,Port);
    2 : SetViewPort(0.5,0.5,1,1,Port);
  end;
  MovePen(InputDateX[1], InputDateY[1]);
  SetColor(Port+10);
  ClearPort(Port);
  for Dot:=2 to MaxDot do
    LinePen(InputDateX[Dot], InputDateY[Dot]);
end; {PlotGraphic}
```

```
Procedure InputDate (PortDate : word);
var
  Dot : word;
begin {Input}
  SetViewPort(0,0,0.5,0.5,3);
  ClearPort(3);
  MoveText(0.3,0.95);
  SetColor(Green);
  OutText('Ввод данных');
  GoToXY(1,15);
  for Dot:=1 to MaxDot do begin
    Read(InputDateX[Dot], InputDateY[Dot]);
  end;
  Sound(320); Delay(300); NoSound;
  with PortArray[PortDate] do
    MaxMin(Port_max_x, Port_min_x, Port_max_y, Port_min_y);
end; {Input}

Procedure PlayMenuMain;
begin {PlayMenuMain}
  SetViewPort(0.5,0,1,0.5,4);
  Pos:=1; ClearPort(Port);
  with MainMenu[1] do begin
    PenX:=0.5; PenY:=0.5; Text:='Выбор окна';
    SetColor(Yellow);
    MoveText(PenX,PenY);
    OutText(Text);
  end;
  SetColor(White);
  with MainMenu[2] do begin
    PenX:=0.5; PenY:=0.4; Text:='Ввод данных';
    MoveText(PenX,PenY);
    OutText(Text);
  end;
  with MainMenu[3] do begin
    PenX:=0.5; PenY:=0.3; Text:='Черчение графика';
    MoveText(PenX,PenY);
    OutText(Text);
  end;
  with MainMenu[4] do begin
    PenX:=0.5; PenY:=0.2; Text:='Выход';
    MoveText(PenX,PenY);
    OutText(Text);
  end;
end; {PlayMenuMain}

Procedure PlayMenuPort;
begin {PlayMenuPort}
  SetViewPort(0.5,0,1,0.5,4);
  Pos:=1;
  ClearPort(Port);
```

```
    with PortMenu[1] do begin
      PenX:=0.5; PenY:=0.5; Text:='Окно 1';
      SetColor(Yellow);
      MoveText(PenX,PenY);
      OutText(Text);
    end;

    SetColor(White);

    with PortMenu[2] do begin
      PenX:=0.5; PenY:=0.4; Text:='Окно 2';
      MoveText(PenX,PenY);
      OutText(Text);
    end;
  end; {PlayMenuPort}

  Function Selector : MainSelect;
  var
    Key : word;
  begin {Selector}
    repeat
      if KeyPressed then begin
        Key:=Ord(ReadKey);
        case Key of
          Up : if Pos <> 1 then begin
            with MainMenu[Pos] do begin
              SetColor(White);
              MoveText(PenX,PenY);
              OutText(Text);
            end;

            Pos:=Pos-1 ;

            with MainMenu[Pos] do begin
              SetColor(Yellow);
              MoveText(PenX,PenY);
              OutText(Text);
            end;
          end;
          Dn : if Pos <> 4 then begin
            with MainMenu[Pos] do begin
              SetColor(White);
              MoveText(PenX,PenY);
              OutText(Text);
            end;

            Pos:=Pos+1;

            with MainMenu[Pos] do begin
              SetColor(Yellow);
              MoveText(PenX,PenY);
```

```
              OutText(Text);
            end;
          end;
        end;
      end;
    until Key = RETURN;
    Selector:=MainSelect(Pos-1);
  end; {Selector}

  Procedure SelectPort(var Port : word);
  var
    Key : word;
  begin {SelectPort}
    Port:=1;
    repeat
      if KeyPressed then begin
        Key:=Ord(ReadKey);
        case Key of

          Up : if Port <> 1 then begin
            with PortMenu[Port] do begin
              SetColor(White);
              MoveText(PenX,PenY);
              OutText(Text);
            end;
            Port:=Port-1 ;
            with PortMenu[Port] do begin
              SetColor(Yellow);
              MoveText(PenX,PenY);
              OutText(Text);
            end;
          end;

          Dn: if Port <> 3 then begin
            with PortMenu[Port] do begin
              SetColor(White);
              MoveText(PenX,PenY);
              OutText(Text);
            end;
            Port:=Port+1 ;
            with PortMenu[Port] do begin
              SetColor(Yellow);
              MoveText(PenX,PenY);
              OutText(Text);
            end;
          end;
        end;
      end;
    until Key = RETURN;
  end; {SelectPort}

  end.
```



Clipper® 5.0 на Российских просторах

Став зарегистрированным пользователем, Вы сможете:

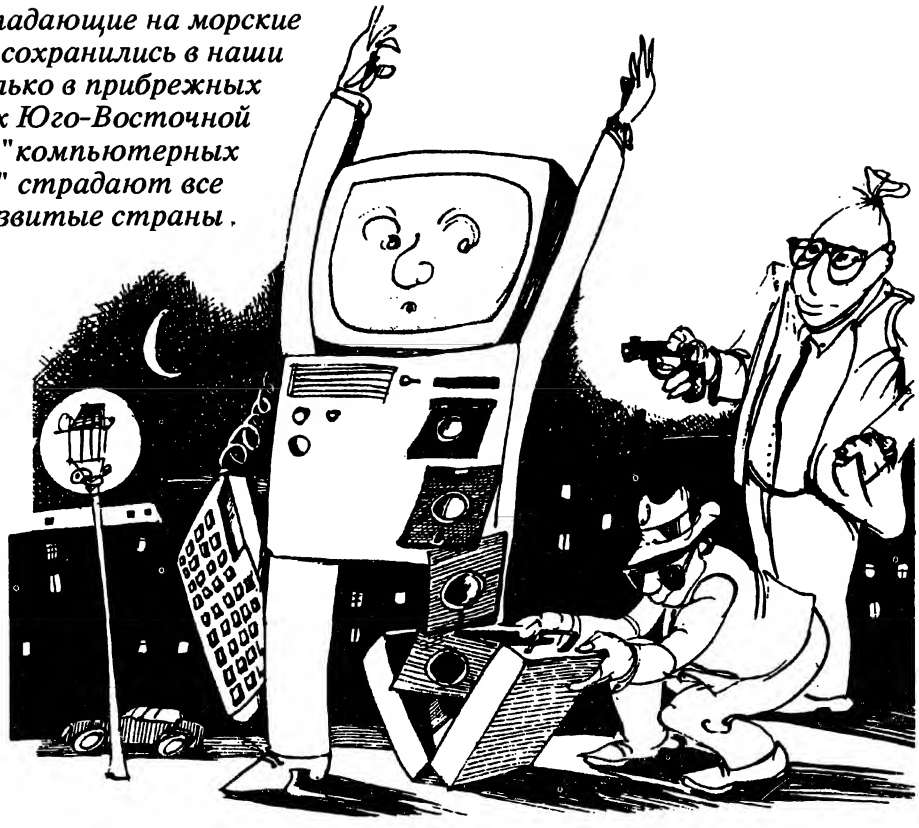
- **обладать полностью русифицированным продуктом и документацией (за рубли);**
- **легально разрабатывать и распространять собственные коммерческие продукты на Clipper;**
- **получать бесперебойные консультации по "горячей" телефонной линии (095) 289-43-00.**

Легальная копия - входной билет в мировое сообщество Clipper

За информацией обращайтесь в СП "Магнит"
Наш адрес: СССР, 127018, Москва, 2-я Ямская, 15.
Телефон: (095) 289-44-77 Факс: (095) 289-34-69

Nantucket®

Пираты, нападающие на морские суда, сохранились в наши дни только в прибрежных водах Юго-Восточной Азии. От "компьютерных пиратов" страдают все развитые страны.



Юридическая защита программ

М. Корсаков

Я забыл сообщить,что первое мое государственное мероприятие ... заключалось в том,что я основал бюро патентов,ибо я знал,что страна без бюро патентов и без хороших законов, защищающих права изобретателей,подобна раку,который может двигаться только вбок или назад.

М.Твен."Янки из Коннектикута при дворе короля Артура".

Хотя оборотистый янки и ограничился заботой о судьбе изобретений, его слова справедливы и в том случае, когда речь идет о других объектах интеллектуальной собственности. Это понятие включает в себя "авторское и другие права, относящиеся к интеллектуальной деятельности в области производства, науки,литературы и искусства" [1].

Формирование информационного общества, возникновение нового вида ресурсов - информационных - привели к появлению новых социальных отношений, в том числе и гражданско-правовых. Возник вопрос о правовой охране программного обеспечения (ПО). Целью такой охраны является закрепление за автором определенных прав в отношении его продукта, в том числе и права на вознаграждение, и стимулирование тем самым деятельности по разработке ПО. При этом система охраны не должна закрывать охраняемые объекты для широкого потребления.

Известны четыре способа правовой защиты интеллектуальной собственности, потенциально подходящие для защиты ПО.

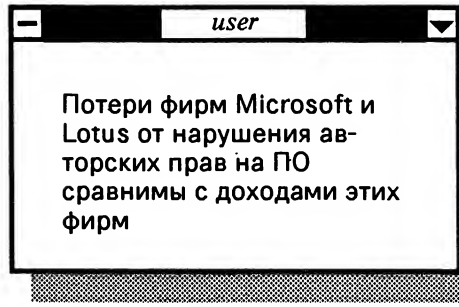
конкретной формы выражения, но не идеи защищаемого объекта.

Во-вторых, система изобретательского (патентного) права, где, наоборот, охраняется принцип, метод частного решения какой-либо задачи.

В-третьих, привлекательна идея специальной охраны, т.е. создание особого законодательства для защиты отдельных видов интеллектуальной собственности с помощью специальных норм. Применительно к ПО такая система позволила бы учесть специфику этого вида интеллектуальной собственности и обеспечить охрану всех элементов программы, что заведомо невозможно в рамках авторского или изобретательского права. Однако препятствием на пути адаптации такой системы является то обстоятельство, что, приняв ее, страна выпадает из мировой системы охраны ПО с помощью авторского права и получает односторонние преимущества по сравнению с другими странами, что не может не вызвать ответные санкции с их стороны. Такими санкциями могут быть создание протекционистских барьеров на пути проникновения ПО из страны со специальной охраной в остальной мир и выключение ее из международного обмена ПО, что отрицательно скажется на конкурентоспособности программных продуктов такой страны.

И, наконец, столь выгодная на первый взгляд идея универсального законодательства об охране интеллектуальной собственности. При этом охране подлежат алгоритмы, вычислительные формулы, принципы, лежащие в основе программы и ее конкретная форма выражения. Однако приверженцы подобного способа часто упускают из вида различия между плодами интеллектуального труда, которые охраняются в рамках разных правовых институтов. Эти различия состоят в большей или меньшей доле субъективности в результатах творческой деятельности. С этой точки зрения компьютерные программы в высшей степени трудно поддаются какой-либо классификации. Мы говорим о сложной структуре программы как результате творческой деятельности.

Вычислительные программы ближе всего к математическим формулам, не входящим в перечень охраняемых объектов. Игровые же скорее похожи



на произведения искусства, так как дизайн в них играет главную роль, а алгоритм как правило общеизвестен. Соотношение творческого и объективного начала в программе определяет способ ее охраны при универсальной системе.

Возможен смешанный вариант охраны ПО, при котором к отдельным компонентам ПО применяются нормы различных институтов права. Например, форма выражения конкретной программы охраняется по нормам авторского права, а лежащий в основе ее алгоритм - с помощью патентного права. Кроме того, возможно применение норм административного права (для регулирования круга лиц и порядка доступа к машинной информации в учреждениях и организациях) как дополнительного инструмента охраны.

Как определить понятие "компьютерная программа"

В настоящее время в странах с высоким уровнем компьютеризации для правовой охраны программного обеспечения используется авторское право.

При этом важно прежде всего определить понятие охраняемого объекта, т.е. компьютерной программы. Прямое указание ее в перечне охраняемых объектов встречается в "Законе об авторском праве" США и "Основах гражданского законодательства СССР и союзных республик". В остальных странах различные виды программ относятся к тем или иным традиционно охраняемым категориям (обычно к литературным и научным произведениям), а само определение дается в разъяснительных документах различных органов. Необходимым условием для установления охраноспособности считается наличие признака творчества и объективной формы выражения (т.е. программа должна быть материально оформлена, например, записана на дискете). Обычно в определении программы присутствуют слова : "... набор команд и переменных, используемых прямо или косвенно в компьютере с целью получения определенного результата...". Дополнительно определяются понятия "описание программы" и "вспомогательный материал" [2]. Заметим, что два этих понятия относятся к числу традиционно охраняемых авторским правом объектов, а именно - к научно-технической литературе.

Однако, хотя на современном этапе программы относятся большинством специалистов к разряду научных произведений, вопрос этот представляется чрезвычайно сложным. Действительно, дать исчерпывающее определение программы невозможно из-за огромного разнообразия программ и постоянного расширения сферы применения компьютеров, ведущего к появлению новых категорий программных продуктов. Таким образом, при распространении норм авторского права на ПО, этот термин необходимо трактовать как можно шире, не давая конкретного определения, чтобы исключить риск устаревания последнего. Оригинальность программы, т.е.наличие творческого начала при ее разработке, не требует проверки по каким-либо качественным или эстетическим критериям. Единственное условие - охраняемая программа

Для компании Lotus Development,Inc недавний выигрыш дела о нарушении авторских прав послужил своеобразным призывом к оружию.Вскоре после привлечения к ответу Paperback Software International,Inc в Бостонском федеральном суде ,в 1990 году Lotus предъявила иск компании Borland International,Inc.

Для Borland это явилось серьезной проблемой.Основанная в 1983 году,эта калифорнийская компания в 1990 году стала конкурировать с такими монстрами в области ПО,как Microsoft,Lotus,Ashton Tate.

Предмет иска,продукт Borland Qattro, явился лишь потенциальной угрозой для Lotus,незначительно подорвав положение этой фирмы на рынке.Этого оказалось достаточно,чтобы разбудить "массачусетского гиганта".

В случае проигрыша Borland должен будет убрать свой продукт с рынка.Более того,по оценкам Dean Witters Reynolds,Inc "это может отпугнуть клиентов".

В случае выведения Quattro с рынка прибыли Borland значительно упадут. Ожидаемый уровень их от продаж этого продукта составляет около 15% от 125 млн. долл. ,полученных в этом году,по оценкам International Data Corp. Разумеется,остаются еще и другие разработки,но "Quattro задумывался как наиболее перспективный товар",- по словам экспертов из Infocorp,компании по исследованию рынка. "Это может привести к значительному ухудшению позиций фирмы".Уолл-стрит отреагировал немедленно:акции Borland упали на 23%.

Однако у фирмы есть еще надежда,основывающаяся на четком различии между данным процессом и иском к Paperback Sostware International,Inc.

Программа Paperback, VP-Planner,почти не отличалась от Lotus 1-2-3 и содержала тот же набор меню и команд. Quattro же,наоборот,не похож на 1-2-3. Дизайн программы совершенно иной,с присущими только ей оригинальными чертами, которым отдают предпочтение пользователи.

Quattro предоставляет большие возможности взаимодействия с пользователем - свойство,способное переманить клиентов у Lotus. В своей рекламе компания упирает на то,что потребителям не нужно осваивать новую систему."Это важное преимущество для завоевания рынка потребителей 1-2-3",-считают эксперты.

Так что, при решении дела в пользу Lotus, ,объем продаж Borland International серьезно снизится.

В пользу Borland говорит и тот факт,что Quattro был выведен на рынок давно, но Lotus до сих пор воздерживалась от предъявления иска.Таким образом, действия этой фирмы могут классифицироваться как нарушение антитрестовского законодательства (прецедент имел место в Сан-Франциско,где суд решил,что Apple Computer,Inc не нарушает права на графическое ПО фирмы Xerox Corp.). Lotus возбудил аналогичное дело против Santa Cruz Operation,Inc,выпустившей более дешевый продукт типа 1-2-3,но все же четко от него отличающийся (он работает в UNIX, а не в DOS, как 1-2-3).

Президент Borland Филипп Р. Канн рассматривает возбуждение дела как попытку Lotus ограничить конкуренцию: "Это не остановит нас на пути завоевания рынка".

не должна быть копией какой-нибудь другой. Оригинальной считается и программа,созданная в результате "творческой переработки" уже существующей (на практике достаточно внести в программу более или менее значительные изменения, касающиеся ее формы выражения, и доказать, что вы "не приложили творческого труда" при ее создании, становится довольно-таки трудно) или "творческой компиляции" при условии демонстрации квалификации и приложении достаточно большого труда (та или иная компоновка обычных подпрограмм, к примеру, не считается нарушением авторского права). Переставив несколько блоков, по-другому скомпоновав информацию при создании базы данных, вы можете спать спокойно, так как понятие оригинальности трактуется чрезвычайно широко.

Автор и его права

Очень важно четкое определение лица, признаваемого автором, и его прав. Известно несколько вариантов, отраженных в законодательстве.

1. Программу создает одно лицо индивидуально. В таком случае оно и считается автором. Если же в работе участвовали несколько человек, права на данную программу носят коллективный характер с сохранением прав отдельных лиц на ее компоненты.

2.Программа разработана лицом по договору. Создатель программы обладает всеми правами на нее, если в договоре не указано иное.

3.Если программу написал наемный работник, действующий в рамках служебных обязанностей, права обычно закрепляются за нанимателем при отсутствии в контракте других условий.

Автор получает целый комплекс прав на свою программу. К числу наиболее важных имущественных прав относятся:

- право на использование или хранение в ЭВМ программы (копии);
- право на изготовление копий;
- право на распространение программы (путем продажи, другой передачи прав собственности, а также сдачи в аренду, напрокат);
- право на переработку (трансляция, адаптация);
- право на вознаграждение.

Существует ряд так называемых личных неимущественных прав (на имя, на неприкосновенность произведения, на опубликование его). Возможна передача автором своих прав как имущественных, так и личных неимущественных другому лицу. Причем, если в Великобритании и США передаваться могут все права, включая право на имя, в континентальной Европе право на имя передаваться не может. Для распространения и использования программы требуется письменное согласие автора. Без такого документа авторские права считаются нарушенными и автор может обратиться в суд.

Исключение составляют несколько случаев, когда использование программы без лицензии автора не носит коммерческого характера. Например, допускается адаптация или создание архивных копий программы, если это необходимо для использования данной программы в данном компьютере. Архивные копии должны

уничтожаться с прекращением права собственности на копию программы. Распространение не должно выходить за определенные рамки (например, должно быть не более одной копии для библиотек; должно быть ограничено тиражирование в целях обучения).

В случае нарушения своих прав, как уже говорилось, автор может обратиться в суд. При рассмотрении иска он должен, во-первых, доказать свое право на программу (с этой точки зрения интересен опыт США, где распространена практика - но не обязательное требование - регистрации объектов авторского права; у нас можно вспомнить о Государственном Фонде алгоритмов и программ как о средстве доказательства). Затем необходимо доказать факт нарушения своих прав. Можно выделить два случая. Первый - когда уже созданный продукт копируется один к одному и продается без лицензии; здесь важно поймать вора с поличным, например, следя за рекламными объявлениями дистрибьюторов ПО. Второй - когда продукт копируется еще в стадии разработки или когда в существующий продукт вносятся незначительные изменения. В этом случае важно доказать, что ответчик имел доступ к работе истца, и что оба продукта значительно совпадают. Нарушитель возмещает автору реальный убыток (выразившийся в падении объема продаж, уменьшении рыночной ценности продукта) и упущенную выгоду (которая определяется на основании валового дохода ответчика за вычетом части не связанной с нарушением авторских прав; эта часть доказывается ответчиком). Автор также может потребовать изъятия и уничтожения всех незаконно изготовленных копий и полуфабрикатов.

И все же, несмотря на казалось бы эффективную и опробованную систему защиты, незаконное использование программных средств представляет значительную проблему. От "компьютерных пиратов" страдают в большей или меньшей степени все развитые страны. Такие крупные поставщики программных продуктов, как Microsoft и Lotus, сообщают о том, что убытки от нарушения авторских прав на ПО сравнимы с доходами этих фирм. Убытки от несанкционированного копирования в Западной Европе составляют примерно 50% от общего рынка программ. Среди компаний и других организаций наблюдается тенденция незаконно копировать ПО и поставлять его вместе с компьютерами. В некоторых фирмах до 90% используемых программных средств - несанкционированные копии. В Италии всего 29.1% пользователей ПК при покупке компьютеров приобретают на законных основаниях также и ПО. Еще хуже ситуация в Испании, где эта цифра составляет 26.7%. Остальные страны более или менее благополучны в этом отношении (Великобритания - 85.4%; Франция - 60.8%;ФРГ - 48.5%). Наиболее страдают от несанкционированного копирования программные средства общего назначения. Основной причиной, по которой компании часто даже поощряют незаконное использование программ, является отсутствие финансовых ресурсов для их законного приобретения. Кроме того, в силу специфики самой компьютерной техники, распространение ПО практически не поддается контролю. Масштабы пиратской деятельности ширятся по мере того, как увеличивается число ПК.

Масштабы пиратской деятельности

Пиратам также помогает и несовершенство существующего законодательства. Стоит лишь раскрыть идею, принцип и немного модифицировать внешний вид, дизайн программы, как вопрос о признании факта нарушения авторского права становится довольно спорным, так как понятие творческая переработка, как уже отмечалось выше, трактуется достаточно широко. Для возбуждения иска (причем на практике это возможно, если он связан с финансовыми претензиями) необходимо четко сформулировать обвинение и представить доказательства нарушения прав автора. Поиск же достаточных для этого улик может вспугнуть пиратов и дать им возможность уничтожить следы преступления. Поощряет кражу программ и сравнительная легкость наказания. Помимо уничтожения незаконных копий и возмещения убытков, предусмотрен штраф (по законам США, например,от 10 до 50 тыс. долл.) или тюремное заключение (обычно от одного до двух лет). По сравнению с прибылью в случае удачи эти меры не столь уж жестки. И, наконец, существуют страны, в которых вообще нет законов об авторском праве или эти законы чрезвычайно мягки и не соблюдаются, но есть ПК и потребители ПО для них. Практически можно обнаружить, что ваши права нарушены, тщательно изучая рекламные объявления фирм-поставщиков ПО (как это делает Microsoft, что позволило этой компании поймать за руку незаконного распространителя ее продуктов - от языка Crosstalk до пакета Lotus 1-2-3 - фирму SKC Tech в Малайзии). Другой способ, особенно эффективный в случае, если ПО незаконно используется в какой-либо организации - проведение контрольных закупок.

Можно также обнаружить вора чисто техническими средствами. Например, при вызове на обслуживание по "горячей линии" с помощью нескольких вопросов можно установить, законно используется программа или нет (для этого достаточно знать регистрационный номер программы; информацию дают и сами трудности - программа может работать только на определенной марке компьютера и, будучи скопированной на неподходящую машину, при обращении за помощью указывает на нарушение авторских прав). Наконец, рекомендуется включать в программы искусственные

ошибки, не влияющие на ее работоспособность, но позволяющие четко идентифицировать данный продукт. Эффективна также техническая защита от копирования (copy protection) с помощью различных встроенных программ.

Еще одной попыткой защитить себя от воровства явилось создание разнообразных ассоциаций, ставящих своей целью борьбу с несанкционированным копированием, воспитание пользователей. Ввиду сложности контроля за распространением ПО особую роль здесь играют неформальные отношения. Большинство профессионалов и грамотных пользователей, уважая чужой труд, не позволят себе незаконное использование программ.

Действенным средством, удерживающим от несанкционированного копирования, могут стать и неблагоприятные последствия для имиджа фирмы в случае огласки.

Компании - члены таких ассоциаций в качестве одной из мер борьбы просто информируют администрацию фирм-нарушителей и сообщают им о юридических последствиях незаконного использования программ. Подобные организации добиваются значительных успехов. По инициативе Британской ассоциации против несанкционированного использования программных средств (FAST), к примеру, было возбуждено свыше 20 исков. В 1988 году выявлены незаконно скопированные средства на сумму более 1 млн ф.ст. Аналогичные ассоциации действуют в США (SPA), ФРГ (VSI), Италии (SIAE). Сопровождение и предоставление услуг по вызовам, льготные условия для университетов (как это делает Word Perfect GMbH со своей "кампус-лицензией") и постоянных крупных клиентов могут помочь в предотвращении незаконного копирования. Распространена также защита алгоритмов и программ в виде "know how". Однако продажа такой программы будет означать и передачу прав собственности на нее.

Основы гражданского законодательства
Союза ССР и республик

РАЗДЕЛ IV

АВТОРСКОЕ ПРАВО

СТАТЬЯ 134. Произведения,на которые распространяется авторское право.

2. К объектам авторского права относятся ... программы для ЭВМ, сборники (...базы данных).

СТАТЬЯ 135. Права автора.

2. Автору произведения принадлежит исключительное право на свое произведение, включающее:

- право авторства;
- право на имя;
- право на неприкосновенность произведения;
- право на опубликование произведения;
- право на использование произведения...;
- право на вознаграждение за разрешение использовать и использование произведения.

СТАТЬЯ 137. Срок действия авторского права.

1. Авторское право действует в течение всей жизни автора и пятьдесят лет после его смерти, считая с 1 января года, следующего за годом смерти автора.

СТАТЬЯ 139. Авторский договор.

1. Использование произведения автора другими лицами ... осуществляется на основании авторского договора.
2. Условия заключенного с автором договора, ухудшающие его положение по сравнению с положением, установленным в законодательстве,не имеют юридической силы и заменяются условиями, установленными законодательством

СТАТЬЯ 140. Служебные произведения.

Авторское право на произведение, созданное в порядке выполнения служебного задания (служебное произведение), принадлежит его автору.

Право использования служебного произведения способом, обусловленным целью задания и в вытекающих из него пределах, принадлежит лицу,по заданию которого создано произведение...

По истечении трех лет с момента представления произведения,а при согласии работодателя - и ранее, права автора на использование произведения и на получение авторского вознаграждения принадлежат ему в полном объеме.

Право автора использовать служебное произведение способом, не обусловленным целью задания, не ограничивается.

Как же обстоят дела с защитой программ в нашей стране

Как же обстоят дела с защитой программ в нашей стране? До недавних пор все ограничивалось теоретическими спорами и проектами законодательства о защите ПО. В достаточно большом количестве работ на эту тему обсуждались все упомянутые выше способы охраны. В действительности же отношения по созданию, распространению и использованию ПО регулировались многочисленными Положениями, инструкциями, Типовыми договорами, почти не рассматривающими вопросы защиты прав автора. Лишь в некоторых из них предусматривались ничтожные санкции "за внесение изменений в программу или передачу третьему лицу " [3] или в графе "Прочие условия" оговаривалась возможность распределения прав. Не было законодательного определения охраняемого объекта, хотя в некоторых актах и различались сама программа и техническая и эксплуатационная документации. Не учитывалась специфика прав автора приме-

нительно к ПО. Ни о каком справедливом вознаграждении автора и речи быть не могло, так как работа оплачивалась, исходя из калькуляции, по тарифам, а не в зависимости от коммерческого успеха программы. Вообще в условиях жесткого регулирования вряд ли можно было говорить о коммерческом распространении программных продуктов.

Кроме того, ввиду отсутствия должной законодательной защиты, на несанкционированное копирование и адаптацию западного ПО на государственном уровне смотрели сквозь пальцы (несмотря на то, что в 1973 году СССР присоединился к Бернской конвенции, положения которой вступили для него в силу с 1975 года и предусматривают охрану ПО, относя его к категории литературных произведений).

Растущее число персональных компьютеров в стране, большая открытость, распространение принципов рынка обусловили актуальность проблемы защиты программ. Западные фирмы-поставщики ПО в первую очередь стремятся обеспечить должный статус своим продуктам на перспективном, но полном опасностей советском рынке. Их поддерживают талантливые профессионалы-программисты, которым поток пиратских копий угрожает потерей заработка. Активно добиваются защиты своих товаров и дистрибьюторы западных программных продуктов, имеющие официальные договоренности с фирмами, и организации, разрабатывающие оригинальные конкурентоспособные программы. Их деятельность схожа с деятельностью различных ассоциаций-производителей ПО, о которых мы говорили выше, и состоит скорее в воспитании пользователей, создании соответствующего общественного мнения; они также выступают активными сторонниками разработки законодательной охраны ПО.

С принятием новых "Основ гражданского законодательства СССР..." [4], согласно которым программы и базы данных включены в перечень охраняемых объектов, наша страна определенно вступила на путь защиты ПО по нормам авторского права. Первый шаг сделан. Однако крайне легкомысленно было бы думать, что этого достаточно. Предстоит, во-первых,согласовать вопрос о защите ПО с независимыми республиками, разработать достаточно эффективный механизм судебной защиты (эту функцию, как представляется, возьмут на себя арбитражи разных уровней). Совершенно необходимо определить основные положения авторского договора на создание и использование ПО, распределяющего имущественные и личные неимущественные права между автором и заказчиком. Такой договор должен иметь законодательные основания для признания его пунктов действительными. Он может содержать следующие положения.

Что делать?

Совершенно необходимо определить основные положения авторского договора на создание и использование ПО, распределяющего имущественные и личные неимущественные права между автором и заказчиком.

- Указание на то, что автор разрабатывал какую-либо идею, на основе которой он создает данный продукт.
- Распределение имущественных и личных неимущественных прав между сторонами, условия прекращения договора и распределение прав в этом случае.
- Обязательство заказчика использовать работу в определенном объеме (например, реализовать не более оговоренного числа копий).
- Порядок расчетов (в зависимости от коммерческого успеха разработки).
- Обязательства сторон по охране предмета договора от расхищения другими лицами.

В той или иной степени некоторые из этих положений отражены в законодательстве и нормативных актах. Проблема состоит в том, что необходимо охватить весь спектр отношений по созданию и использованию ПО и установить принципы справедливого вознаграждения авторов, а также создать механизм защиты.

Кроме того, необходимо учитывать специфику ПО в случае продажи лицензии на распространение (в том, что касается адаптации, архивных копий, субаренды). Естественно, что законодательство устанавливает лишь общие принципы охраны. Использование договора в каждом конкретном случае позволяет более гибко регулировать отношения по поводу ПО, снижает риск незаконного копирования программ.

Литература

1. Советский энциклопедический словарь. М., 1989, стр.500.
2. Типовые положения по охране программного обеспечения вычислительных машин. Разработаны консультационной группой ВОИС, 1977г.
3. Типовой договор на оказание научно-технических услуг потребителям программных средств вычислительной техники. Утв. Постановлением Госарбитража СССР от 27.08.85 г.
4. "Ведомости СНД и ВС СССР", 1991, № 26.

COPYRIGHT в вашей программе

В.Водолазкий

Спасение утопающих-дело рук самих утопающих!

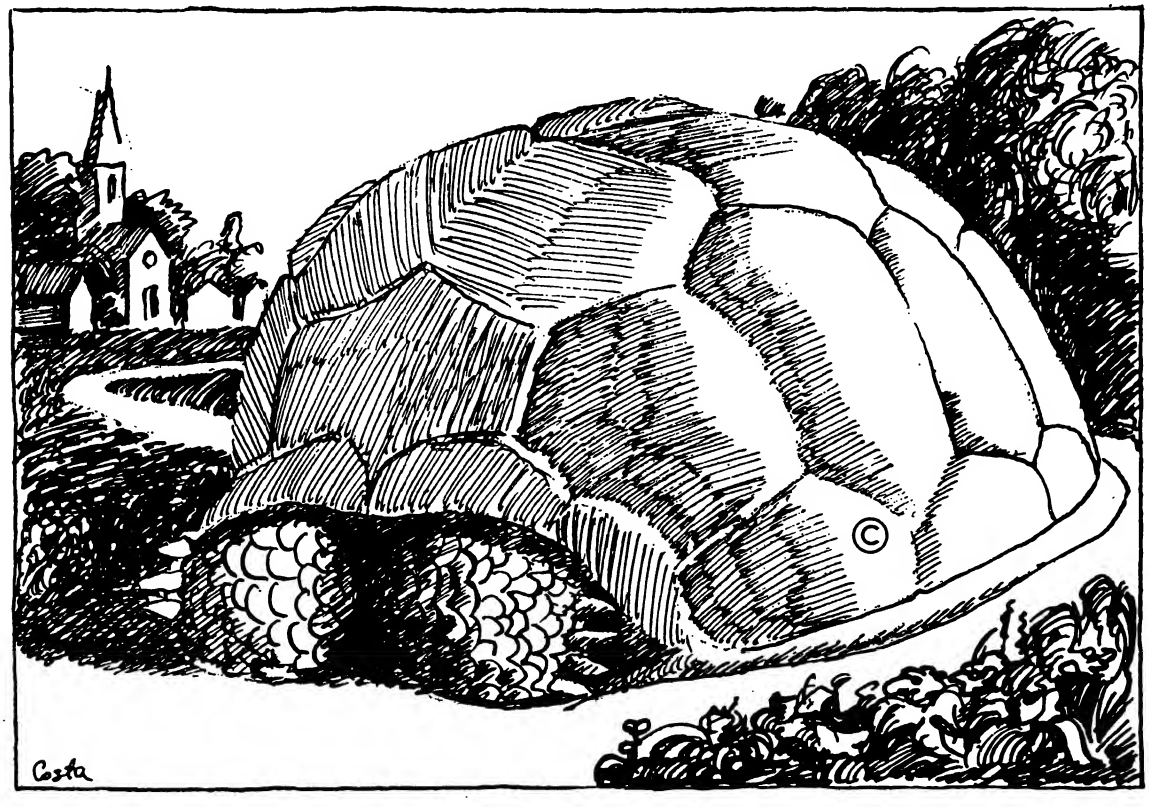
Когда программа закончена, у автора возникает вполне законное желание поставить свой автограф, чтобы восхищенные (или разъяренные) пользователи знали, кому адресовать свою благодарность (или, соответственно, проклятия). Но ведь не секрет, что периодически у некоторых коммерсантов возникает непреодолимое желание присвоить себе результаты чужого труда, а для ликвидации последних угрызений совести достаточно использовать PCtools или Norton Disk Editor для замены выстраданной метки "Бирюлевский колхоз им. В.И.Лысенко" на "Брокерско-Дилерскую ассоциацию "Кащей".

При этом автору программы, как правило, невозможно "заочно" отстоять свое авторство. В результате не удается даже утешаться лозунгом: "Хороших законов у нас принято немало. Дело за тем, чтобы претворить их в жизнь!" [1]. Ведь у нас все еще нет закона о защите авторских прав, поэтому имеет смысл приложить некоторые усилия для защиты своего авторства и усложнения жизни любителям легкой наживы.

Приведенные в данной заметке процедуры реализованы на различных языках вовсе не с целью запутать читателя. Автор хотел лишь показать, что описанные методы легко реализуются на любом языке от Ассемблера до Бейсика.

Итак, установка и защита автографа (по буржуйски - Copyright) в вашей программе.

Очевидно, что для установки автографа необходимо определить в тексте программы символьную строку и при



Два главных распределительных центра корпорации Revlon простаивали 3 дня, когда корпорация Logisticon вывела из строя компьютеры Revlon, нарушив работу программного обеспечения, которое не было оплачено косметической компанией. Logisticon прекратил в компьютерах товарного склада обработку учета, распоряжений о продажах и распределении продукции, но не причинил вреда файлам с данными.

Такие секреты называют ключами доступа, "черным ходом", ловушками или бомбами замедленного действия. Logisticon - не единственный поставщик, кто вставляет их в программное обеспечение при продаже крупным клиентам. Многие разработчики программного обеспечения - особенно мелкие поставщики - предусматривают возможность произвести односторонние действия в случае невыполнения финансовых обязательств. Но это происходит крайне редко. Если поставщик имеет такой секрет, об этом не знает даже его собственный адвокат.

запуске программы вывести ее на экран дисплея. Недостаток такого простого подхода состоит в том, что данная текстовая строка легко обнаруживается в теле EXE- или COM- файла с помощью контекстного поиска и затем заменяется на любую другую. Как мы можем этому помешать?

На любом языке от Ассемблера до Бейсика...

Во-первых, мы можем использовать контроль целостности строки автографа, например за счет подсчета ее контрольной суммы (пример 1). Как видно из программы, в случае обнаружения несоответствия вычисленного значения эталонному, выполнение программы будет прекращено.

Во-вторых, мы можем хранить строку автографа в теле программы в зашифрованном виде (пример 2). В этом случае просмотр файла с помощью PCtools не позволит обнаружить ваш автограф, и в большинстве случаев защитит вашу программу от посягательств.

...легко реализуются предлагаемые методы шифрования строки автографа

И, наконец, мы можем использовать оба эти способа одновременно. В результате, для того, чтобы "вычистить" из программы ваш автограф, взломщику придется садиться за отладчик типа Turbo Debugger для выяснения логики подсчета контрольных сумм и принципа шифрования, затем выполнить соответствующие преобразования для своей метки, и лишь затем внести изменения в украденную программу, причем не в текстовом, а в шестнадцатиричном режиме. А все это требует как квалификации, так и времени, что в конце концов выражается некоторой денежной суммой, и, таким образом, при грамотном подходе может оказаться, что выгоднее купить программу у Вас, чем платить за замену Вашего автографа.

"Все это здорово, - скажет прагматичный читатель, - но я ведь тоже должен буду высчитывать каждый раз контрольную сумму и шифровать строку автографа". А вот и нет! Для этой цели автор хочет предложить вам небольшую программу Copyr. Программа написана на языке Turbo Pascal (пример 3) и предназначена для генерации процедуры на языке Pascal (пример 4), которая содержит все необходимые данные. После генерации сформированная процедура включается в вашу программу с помощью директивы {$i <имя файла>} (пример 5), а в тело основной программы включается вызов процедуры CopyRight. Для реализации генерации текста программы на других языках в Copyr необходимо внести очевидные изменения.

Приведенный алгоритм шифрования строки автографа может быть заменен на ваш собственный. Например, вы можете шифровать каждый символ строки с помощью операции XOR, использовать битовые перестановки и т.д. Не забудьте только внести соответствующие изменения в строки программы Copyr, которые генерируют код расшифровки строки. Аналогичным образом Вы можете внести изменения и в алгоритм подсчета контрольной суммы.

Литература:

Л.И.Брежнев. Отчетный доклад на XXV съезде КПСС.

; Пример 1. Программа на языке Ассемблер, иллюстрирующая защиту автографа,
; с помощью подсчета его контрольной суммы.

```
        .
        .
        mov     ah,9            ;вызов функции DOS - печать строки
        lea     dx, Copyright   ;загружаем адрес текстовой строки
        push    dx              ;сохраняем его в стеке
        int     21h
        cld                     ;просмотр строки вперед
        pop     si              ;восстановление адреса строки автографа
        mov     cx, copy_length ;длина строки автографа в словах (!)
        xor     bx,bx           ;очистка накопителя суммы
@@1:    lodsw                   ;читаем очередное слово
        add     bx,ax           ;корректируем накопитель
        loop    @@1
        cmp     bx, magic_number ;сравниваем полученную сумму с эталоном
        je      @@2             ;если все в порядке, продолжаем работу
        mov     ax, 4c01h       ;аварийное прекращение работы
        int     21h
@@2:
        .
        .
        .
Copyright       db      'В.В.Водолазкий, (C) 1991 $'
        even                    ;коррекция для работы со словами

copy_length     equ     ($ - copyright) div 2

magic_number    equ     5375h   ;эталонная контрольная сумма
```

Rem Пример 2. Программа на Бейсике, иллюстрирующая хранение автографа
Rem в своем теле в зашифрованном виде.

```
data    $34,$23,$54,$56,$65,$76
data    $11,$23,$72,$60,$12,$10
data    ......
data    ......
restore
locate 1,1
color 7,2
for i = 1 to 20
read a
print chr$(a+i);
next i
return
```

{ Пример 3. Программа Copyr. }

```
Program copyr;

uses CRT;

var
  f : text;
  CopyRight, filename : string;
  V : array [1..80] of byte; {зашифрованный автограф}
  VL : byte; {длина строки автографа}
  W : word; {контрольная сумма автографа}

procedure Crypt; {процедура шифрования строки автографа}
var
  i : byte;
begin
  VL:=Length(CopyRight);
  For i:=1 to VL do
  V[i]:=Ord(CopyRight[i])-i;
end;

Procedure CheckSum; {подсчет контрольной суммы строки автографа}
var
  i : byte;
begin
  W:=0;
  For i:=1 to VL do
    W:=W+ord(CopyRight[i]);
end;

var
  i,j,M : byte;
begin { основная программа }
  TextColor(Yellow);
  Write('Введите имя создаваемого файла : ');
  TextColor(LightGreen);
  Readln(filename);
  Assign(f, filename);
  {$i-}
  Rewrite(f);
  {$i+}
  if IOResult <> 0 then begin
    TextColor(LightRed);
    Writeln;
    Writeln('Ошибка создания файла');
    TextColor(LightBlue);
    Halt(3);
  end;
  Writeln;
  TextColor(Yellow);
  Write('Ваш автограф : ');
```

```
    TextColor(LightGreen);
    Readln(Copyright);
    Crypt;
    CheckSum;
    Writeln(f,'Procedure Copyright;');
    Writeln(f);
    Writeln(f,' { ',CopyRight,' }');
    Writeln(f);
    Writeln(f,'TYPE');
    Writeln(f,' AA = array [1..',VL,'] of byte;');
    Writeln(f);
    Writeln(f,'CONST');
    Writeln(f,' CopyR : AA = (');
    j := 0;
    For i := 1 to VL do
    begin
      If J = 0 then Write(f,' ');
      Write(f,' ',V[i]:4);
      IF i < VL then Write(f,',');
      Inc(j);
      IF j > 6 then
      begin
        Writeln(f);
        J := 0;
      end
    end;
    Writeln(f,' );');
    Writeln(f);
    Writeln(f);
    Writeln(f,' CheckSum = ', W,';');
    Writeln(f);
    Writeln(f,'VAR');
    Writeln(f,' i,j : byte; W : word;');
    Writeln(f);
    Writeln(f,' begin');
    Writeln(f,' W := 0;');
    Writeln(f,' for i := 1 to ',VL,' do');
    Writeln(f,' begin');
    Writeln(f,' j := CopyR[i] + i;');
    Writeln(f,' W := W + J;');
    Writeln(f,' Write(Chr(j));');
    Writeln(f,' end;');
    Writeln(f,' If W <> CheckSum then Halt(1)');
    Writeln(f,' end;');
    Close(f);
end.
```

{ Прим 4. Процедура Copyright }

```
Procedure Copyright;

{ © В.В.Водолазкий }

Type

  AA = array [1..56] of byte;

Const

  CopyR : AA = (
        137, 169, 224, 157, 27, 28, 142,
        152, 216, 226, 21, 156, 19, 135,
        209, 223, 208, 218, 15, 12, 145,
        205, 201, 149, 135, 145, 133, 4,
        5, 110, 143, 141, 135, 192, 139,
        188, 253, 250, 1, 105, 0, 246,
        87, 2, 85, 0, 83, 126, 115,
        124, 120, 108, 114, 116, 113, 113);

  CheckSum = 7853;

VAR
  i,j : byte;
  W : word;

begin
  W:=0;
  for i:=1 to 56 do
  begin
    j:=CopyR[i] + i;
    W := W + J;
    Write(Chr(j));
  end;
  If W <> CheckSum then Halt(1)
end;
```

{ Прим 5. Включение процедуры Copyright в пользовательскую программу. }

```
program CopyrTest;

{$i ris4.txt}

begin
  Copyright
end.
```

"Кто–то программировал и на моем компьютере и стер мой любимый "Тетрис"!"

В. Аношин

Защита программ и данных на IBM PC

Статья рассказывает о программных средствах защиты от несанкционированного доступа к программам и данным

У многих работающих на IBM PC возникают проблемы, связанные с защитой данных. Это объясняется тем, что наиболее часто используемая операционная система - DOS не имеет встроенных средств ограничения доступа к данным, так как она является однопользовательской.

Существует большое число программных средств, позволяющих осуществить защиту данных. Но не всегда удается приспособить имеющиеся средства защиты к конкретным нуждам, не всегда известно, насколько надежную защиту они обеспечивают. К тому же , как только средство защиты получает широкое распространение, для него сразу находится "противоядие". Да и места в памяти машины и на диске эти системы занимают иногда слишком много.

Наверное единственным надежным способом защиты информации является кодирование ее по паролю, вводимому с клавиатуры, или с ключевой дискеты. Человек, не владеющий паролем, никогда не получит доступа к этим данным. Но этот способ страдает одним существенным недостатком - он слишком медленный, ведь перед каждым использованием данных нужно время, чтобы их раскодировать, а

при их изменении, и закодировать. Если с данными работать постоянно, то этот способ неприемлем.

Все другие способы защиты основаны на использовании особенностей работы операционных систем и ЭВМ. Эти способы являются машинно-зависимыми.

В случае с PC-совместимой техникой защита основывается на нестандартном способе записи на магнитный диск, в результате чего операционная система не может получить доступа к данным без помощи специальных программ, контролирующих эти данные. Преимуществом такого подхода является то, что практически не уменьшается скорость доступа к данным, не требуется кодирование данных после их изменения, а значит, на защищенность данных не влияют такие события, как, например, выключение питания во время работы. Такие системы требуют наличия в памяти машины специальных драйверов, отвечающих за доступ. Способ доступа к информации определяется алгоритмом работы такого драйвера. Чем сложнее алгоритм, тем сложнее его "разгадать" и получить доступ к данным в обход драйвера.

Аналогичным образом работает и большинство способов защиты программ от копирования - в программу заносятся какие-либо уникальные характеристики диска и предусматривается проверка этих характеристик при каждом запуске программы. Если она была скопирована, то данные не совпадают, и программа "отказывается" работать.

Такую защиту при необходимости легко обмануть, все дело во времени и в уровне подготовки "взломщика". Вообще, если затраты на "взлом" вашей защиты превосходят ценность защищенной информации, можно считать, что защита достаточно надежна.

Задачу "взлома" защиты можно существенно усложнить, если применять не распространенные пакеты программ, предназначенные для этой цели, а свою собственную защиту. Возможно, она будет менее грозной, но зато ее не удастся сломать стандартными методами.

Создание собственной защиты предполагает наличие знаний о логическом устройстве дисков и порядке загрузки DOS. Рассмотрим логическую структуру диска DOS.

С точки зрения DOS диск состоит из определенного числа логических секторов, нумеруемых с нуля, размером 512 байт. Пространство логических секторов делится на 4 части, как показано в табл. 1.

programmer

Задачу "взлома" защиты можно существенно усложнить, если применять не распространенные пакеты программ, предназначенные для этой цели, а свою собственную защиту

FAT - это таблица распределения файлов, содержащая информацию о физическом расположении файла на диске и о занятости секторов. В FAT хранится также информация о "сбойных" участках диска. Количество занимаемых FAT секторов вычисляется умножением числа секторов в одной FAT на количество копий FAT на диске (обычно 2). Так как FAT располагается сразу за резервными секторами, то их число является смещением FAT от начала диска.

В секторах, отведенных под корневую директорию, хранится информация об именах файлов и директорий, об их размерах, датах создания или модификации, а также ссылки для FAT на начало файлов. Эта информация выводится операционной системой по команде DIR. Размер директории в секторах равен числу элементов в ней, деленному на 16 (число элементов, помещающихся в одном секторе). Смещение директории равно смещению FAT плюс число секторов в FAT- таблицах.

Четвертая часть – это часть данных, которая начинается сразу за корневой директорией и занимает все остальное пространство диска. В этой части и находится вся информация из файлов.

Наибольший интерес для простой защиты диска представляет BOOT- сектор. В нем содержится вся информация о структуре диска и код загрузки ОС. Операционная система при каждом чтении файла проверяет BOOT-сектор для определения местонахождения FAT и корневой директории. Если в BOOT-секторе записана неверная информация, то прочитать данные средствами операционной системы нельзя. Этим фактом можно воспользоваться для целей защиты.

Табл. 1. Структура пространства логических секторов

Загрузочный (BOOT) сектор и резервные секторы	1
2 копии FAT	2
Корневая (ROOT) директория	3
Секторы данных	4

Смещение	Длина	Описание
00	03	Безусловный переход на код загрузки
03	08	Имя компании и версия системы
0BH	02	Число байт на сектор
0DH	01	Число секторов на кластер
0EH	02	Число резервных секторов перед FAT
10H	01	Число FAT
11H	02	Максимальное число 32-байтовых элементов корневого оглавления
13H	02	Общее число секторов на логическом диске
15H	01	Дескриптор носителя (1-й байт FAT)
16H	02	Число секторов в одной FAT
18H	02	Число секторов на дорожке
1AH	02	Число головок чтения-записи
1CH	02	Число спрятанных секторов
1EH	Начало кода и данных загрузки	

Табл. 2. Структура загрузочного сектора диска

Попробуем реализовать на дискете такую защиту, что нельзя будет получить доступ к данным на ней иначе, как загрузив ОС с этой дискеты.

Для правильной интерпретации данных на диске DOS анализирует таблицу параметров диска, находящуюся в начале загрузочного сектора до кода загрузки (табл. 2).

Версии DOS, начиная с 4.00, имеют более развитую таблицу параметров, тем не менее приведенная структура справедлива для всех версий.

BIOS, а в случае с винчестером программа загрузочного сектора винчестера, считывает загрузочный сектор в память и передает на него управление. В задачу кода загрузки входят считывание системных файлов с диска в память и передача им управления. При этом уже не имеет значения, производится загрузка с винчестера или с диска, так как структуры логического диска на винчестере и дискеты ничем не отличаются.

Переместим содержимое загрузочного сектора в другую область диска, а на его место поместим написанный нами код. Без информации о расположении системных областей DOS не сможет прочитать файлы с диска, а если расположить эти области нестандартно, то без дизассемблирования загрузочного кода их вообще нельзя будет прочитать.

Решить эту задачу относительно просто. Достаточно переместить целиком содержимое 0 дорожки в другую часть диска, например, на последнюю дорожку. Это дорожка с номером 39 (для дискет на 720 Кбайт и 1.2 Мбайт - 79), которую DOS редко использует. При загрузке с нашей дискеты мы должны будем переадресовывать обращения к дорожке 0 на последнюю дорожку. Для этого мы воспользуемся приемом вируса STONED. Этот вирус записывает свой код на место загрузочного сектора, а сам сектор прячет в другое место на диске. При загрузке с зараженной дискеты вирус попадает в память до операционной системы, переписывает свой код в самые верхние адреса памяти и изменяет переменную BIOS, содержащую размер памяти, так, что DOS не обращается к этим адресам, считая что их нет в машине!!! После этого вирус перехватывает прерывание BIOS номер 13h (функции работы с дисками) в целях защиты своего кода на диске и размножения. Затем он загружает настоящий корневой сектор и передает ему управление. Наша программа не будет размножаться, зато она будет спрашивать при загрузке пароль.

Текст программы приводится в конце статьи с подробными комментариями.

Защита устанавливается на любую отформатированную дискету, содержащую системные файлы. Установка производится запуском программы с паролем в качестве параметра.

```
; Прим. 1. Программа запроса пароля

.MODEL          TINY
.CODE
        ORG     100H                    ;Стартовый адрес для "com" файла

; *** Программа установки загрузчика на винчестер ***
MAIN    PROC
        PUSH    DS                      ;Сохраним регистр данных
        MOV     AX,CS                   ;Запишем значение CS
        MOV     DS,AX                   ;в DS и ES через AX
        MOV     ES,AX
;Узнаем, какой устанавливать пароль
        MOV     SI,5DH                  ;Адрес пароля в командной строке
        LEA     DI,PASS                 ;(до 8 символов)
        XOR     CX,CX
READ_PASS:
        LODSB
        CMP     AL,20H
        JE      M00
        CMP     AL,40H
        JNA     M0
        CMP     AL,5AH
        JA      M0                      ;Если заглавная буква, то
        ADD     AL,20H                  ;переведем ее в нижний регистр
M0:
        INC     CX                      ;Запишем очередную букву
        STOSB                           ;в код загрузки
        JMP     READ_PASS
M00:
        MOV     LEN_PASS,CX             ;Длина пароля
;Прочитаем корневой сектор
        MOV     DX,00H                  ;Сторона 0 Диск 00h
        MOV     CX,0001H                ;Дорожка 0 Сектор 1
        LEA     BX,BUFFER               ;Адрес буфера ввода
        MOV     AL,1
        CALL    GET_SECT                ;Подпрограмма чтения секторов
;Прочитаем 0 дорожку
        MOV     DX,00H                  ;Сторона 0 Диск 00h
        MOV     CX,0001H                ;Дорожка 0 Сектор 1
        LEA     BX,BUFFER               ;Адрес буфера ввода
        MOV     AL,BYTE PTR trksecs     ;Секторов на дорожку
        CALL    GET_SECT
;Подпрограмма чтения секторов
;Определим номер последней дорожки
        MOV     AX,totsecs
        CMP     AX,800
        JA      M000
        MOV     WORD PTR trk,39
        JMP     SHORT M001
M000:
        MOV     WORD PTR trk,79
M001:
;Запишем дорожку
        MOV     DX,0                    ;Сторона 0 Диск 0
        MOV     CL,01H                  ;Сектор 1
        MOV     CH,BYTE PTR trk         ;Дорожка=trk
        MOV     AL,BYTE PTR trksecs     ;Секторов на дорожку
        LEA     BX,buffer               ;Адрес буфера ввода/вывода
        CALL    PUT_SECT
;Запишем наш код загрузки
        MOV     DX,0                    ;Сторона 0 Диск 0
        MOV     CX,01H                  ;Сектор 1 Дорожка 0
        MOV     AL,1                    ;Секторов 1
        LEA     BX,BOOT                 ;Адрес корневого сектора
        CALL    PUT_SECT
;Испортим FAT и корневую директорию
        MOV     DX,0                    ;Сторона 0 Диск 0
        MOV     CL,02H                  ;Сектор 2
        MOV     CH,00                   ;Дорожка 0
        MOV     AL,BYTE PTR trksecs     ;Секторов на дорожку
        DEC     AL
        MOV     BX,0                    ;Адрес буфера ввода/вывода (произвольный)
        CALL    PUT_SECT
        JMP     EXIT
MAIN    ENDP

; *** Подпрограмма записи сектора на диск ********************
PUT_SECT        PROC    NEAR
        MOV     SI,3                    ;При ошибке повторить 3 раза
M20:    MOV     AH,3                    ;Функция записи
        INT     13H
        JNC     M22                     ;Выйти, если нет ошибки записи
        XOR     AX,AX
```

```
        INT     13H
        DEC     SI
        JNZ     M20
        LEA     SI,ERR_WRT          ;Вывод сообщения об
        CALL    PUTSTR              ;ошибке записи на диск
        JMP     EXIT
M22:
        RET
PUT_SECT        ENDP

;*** Собственно код защиты ***
        ORG     200H
BOOT    PROC    NEAR
        DB      0EAH
        DW      OFFSET M1           ;jmp far 07A0h:M1
        DW      07A0h
D1:     DB      '(C) 1991 BY ANOSHIN SLAVA.',0DH,0AH
        DB      'ENTER PASSWORD:',0
ERR_RD:
        DB      0DH,0AH,'SECTOR READING ERROR',0
trk:    DW      40                  ;Спрятанная дорожка
M1:
        XOR     AX,AX               ;Установим регистры
        MOV     DS,AX               ;сегментов
        MOV     SS,AX
        MOV     SP,7C00H
        MOV     AX,DS:413H          ;AX= размер оперативной памяти
        DEC     AX                  ;Уменьшим память на
        DEC     AX                  ;2 Кбайта
        MOV     DS:413H,AX
        MOV     CL,6                ;Вычислим сегмент
        SHL     AX,CL               ;'спрятанной' памяти
        MOV     ES,AX               ;Скопируемся в эту память
        MOV     DI,200H
        MOV     SI,7C00H
        MOV     CX,DI
        CLD
REP     MOVSB
        MOV     BX,7C01H
        MOV     [BX],OFFSET CONT
        MOV     AX,ES
        MOV     [BX+2],AX
```

```
        JMP     BOOT                ;Передадим управление на
                                    ;скопированный код
CONT    LABEL WORD
        MOV     AX,ES
        MOV     DS,AX
        LEA     SI,D1               ;Адрес запроса на пароль
        CALL    PUTSTR              ;Выведем запрос на пароль
GETPASS:
        MOV     DI,400H             ;Адрес буфера ввода-вывода
GETSTR:
        XOR     AH,AH               ;ah=0 Функция чтения символа
        INT     16H                 ;с клавиатуры без отображения
        STOSB                       ;его на экране
        CMP     AL,0DH              ;Нажат "ввод" ?
        JNE     GETSTR              ;Повторять, пока не нажат "ввод"
        MOV     DI,400H             ;Введенный пароль
        LEA     SI,PASS             ;Истинный пароль
        MOV     CX,LEN_PASS
REPE    CMPSB                       ;Сравним пароль
        JNE     GETPASS             ;Если не тот - вводим снова
;Установим свой обработчик преывания 13h
        XOR     AX,AX
        MOV     ES,AX               ;Сегмент 0
        MOV     BX,13h*4            ;Дескриптор прерывания 13h
        MOV     AX,ES:[BX]          ;AX=смещение обработчика
        MOV     DX,ES:[BX+2]        ;DX=сегмент
        MOV     _INT_13H,AX         ;Запомним адес старого
        MOV     _INT_13H[2],DX      ;обработчика прерывания
        MOV     ES:[BX],OFFSET INT_13H;Пишем адрес своего
        MOV     AX,CS               ;обработчика
        MOV     ES:[BX+2],AX        ;Прочитаем настоящий загрузчик систе-
                                    мы
        MOV     BX,7C00h
        XOR     DX,DX               ;Диск 0 Сторона 0
        MOV     AX,1                ;Секторов 1
        MOV     CL,1                ;Сектор 1
        MOV     CH,0                ;Дорожка 0!!!!
        CALL    GET_SECT
        DB      0EAH                ;Передадим управление
        DW      7C00H,0000H         ;загрузчику системы
                                    ;(jmp far 0000:7c00)
BOOT    ENDP
```

```
;*** Обаботчик перывания BIOS 13h ***
_INT_13H        DW      0,0
INT_13H         PROC NEAR
        PUSH    DS
        PUSH    CS
        POP     DS
        CMP     AH,2
        JB      CALLER
        CMP     AH,3
        JA      CALLER
        CMP     DX,0
        JNE     CALLER
        CMP     CH,0
        JNE     CALLER
        MOV     CH,BYTE PTR trk
CALLER:
        PUSHF
        CALL    DWORD PTR _INT_13H
        POP     DS
        IRET
INT_13H         ENDP

;*** Подпрограмма вывода строки на экран. Адрес строки - в регистре si ***
PUTSTR          PROC NEAR
        LODSB                           ;Загрузить в al выводимый символ
        CMP     AL,00H                  ;0 - конец строки
        JE      M8
        MOV     BX,7
        MOV     AH,0EH                  ;Функция вывода символа на экран
        INT     10H
        JMP     SHORT PUTSTR
M8:
        RET
PUTSTR          ENDP
LEN_PASS        DW      0000H           ;Длина пароля
PASS            DB      20 DUP(0)       ;Пароль

;*** Подпрограмма чтения сектора с диска ***
GET_SECT        PROC NEAR
        MOV     SI,3                    ;При ошибке повторить 3 раза
M10:
        MOV     AH,2                    ;Функция чтения сектора с диска
        INT     13H
        JNC     M12                     ;Выйти, если нет ошибки чтения
        XOR     AX,AX
        INT     13H
        DEC     SI
        JNZ     M10
        LEA     SI,ERR_RD               ;Вывод сообщения об
        CALL    PUTSTR                  ;ошибке чтения с диска
        JMP     EXIT
M12:
        RET
GET_SECT        ENDP

;*** Подпрограмма выхода ***
EXIT            PROC
        POP     DS
        MOV     AX,4C00H                ;Функция выхода из программы
        INT     21H                     ;Прерывание DOS
EXIT            ENDP
.DATA

;*** Область данных ***
ERR_WRT         DB      "SECTOR WRITING ERROR",0

buffer          label word

;Структура таблицы параметров диска
                db      3 dup(?)
nam             db      8 dup(?)        ;Имя компании и версия системы
sectsiz         dw      512             ;Число байт на сектор
clastsiz        db      2               ;Число секторов на кластер
ressecs         dw      1               ;Число резервных секторов перед FAT
fatcnt          db      2               ;Число FAT
rootsiz         dw      112             ;Максимальное число элементов оглавления
totsecs         dw      720             ;Общее число секторов на логическом диске DOS
media           db      0FDh            ;Дескриптор носителя (1-й байт FAT)
fatsize         dw      2               ;Число секторов в одной FAT
trksecs         dw      9               ;Число секторов на дорожку
headcnt         dw      2               ;Число головок чтения-записи
hidncnt         dd      0               ;Число спрятанных секторов

        END     MAIN
```

Сленг программистов–хэкеров Америки

HACKERESE
- язык хэкеров, программистский жаргон

HAIR
- трудоемкая, сложная работа

GROVEL
- рыскать (без видимого результата, например при просмотре файлов);
- штудировать (например документацию на систему)

GRIND
- придавать (программе) эстетический вид (располагая надлежащим образом строки листинга), шлифовать (программу);
- перемалывать, многократно прокручивать (бесполезную задачу)

FROBNICATE
- бесцельно манипулировать (например клавишами на пульте)

FUDGE
- "состряпанная наспех" программа;
- подогнать под ответ

ZORCH
- работать с огромной скоростью;
- мгновенно продвигать, прокручивать информацию (напр. через сеть)

WOW
- восклицательный знак (название символа)

WORM
- тире (название символа)

QUES
- вопрос (название символа)

WALLPAPER
- длинная распечатка, "простыня"

SOS
- вычесть единицу, уменьшить на единицу

AOS
- добавить единицу, увеличить на единицу

KLUDGE
- клудж (устройство, программа или часть программы, которые теоретически не должны работать, но почему-то работают);
делать клудж (в программе), вставлять клудж (в программу);
- "ляп" (в программе)

JOCK
- программист, пишущий программы "в лоб"; нетворчески;
- жокей

DWIM
- ненужная добавка (усложняющая программу)

DRAGON
- дракон (системная программа, периодически выполняющая служебные функции незаметно для пользователя)

DIKE
- удалять, заглушать (например дефектную часть программы)

CROCK
- хрупкая, неустойчивая программа (боящаяся изменений);
- громоздкая конструкция, громоздкая структура, "монстр"

CATATONIA
- состояние безжизненности системы

BUZZ
- "зависать";
- "жужжать" (об ЭВМ, работающей в коротком цикле)

BUM
- "улучшать" (например программу, ценой потери ее четкости);
- мелкое "улучшение" (напр. программы, обычно лишнее)

Интерфейсы – путь к взаимопониманию

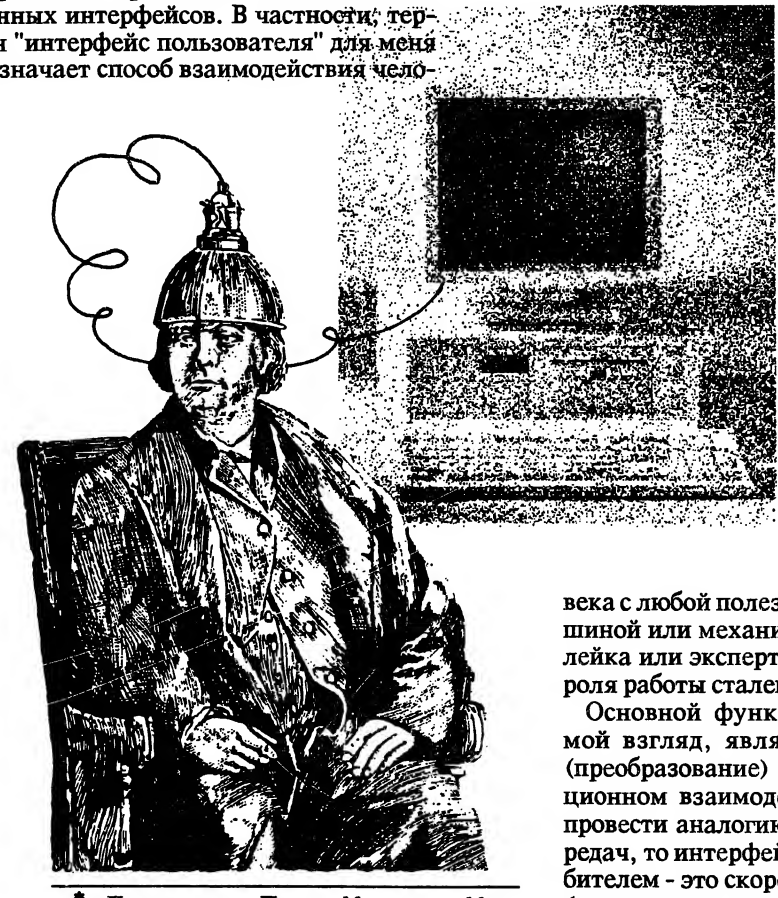
Обзор – размышление *

А. Свириденко

1. Средство или способ

Словарь по программированию и информатике А.Б.Борковского определяет понятие "интерфейс" как "совокупность средств и правил, обеспечивающих логическое или физическое взаимодействие устройств и/или программ вычислительной системы". Это наиболее общее из всех определений "интерфейса", которые мне удалось найти, и все же оно демонстрирует несколько ограниченное понимание, к тому же - весьма техническое.

Итак, сущность понятия "интерфейс" определяется сегодня как "средство для взаимодействия". Однако, как мне кажется, можно трактовать этот термин более широко - как способ взаимодействия любых систем. Таким образом, интерфейс отвечает за то, КАК происходит взаимодействие, а не за то, ЧТО ОБЕСПЕЧИВАЕТ это взаимодействие. Именно в этом смысле мне и хотелось бы поразмышлять о современных подходах к разработке и реализации человеко-машинных интерфейсов. В частности, термин "интерфейс пользователя" для меня обозначает способ взаимодействия человека с любой полезной ему системой, машиной или механизмом, будь то садовая лейка или экспертная система для контроля работы сталеплавильного завода.



Основной функцией интерфейса, на мой взгляд, является трансформация (преобразование) знания при информационном взаимодействии систем. Если провести аналогию с линией электропередач, то интерфейс между ГЭС и потребителем - это скорее не провода, а трансформаторные станции, обеспечивающие понижение напряжения.

* - По результатам Первого Московского Международного семинара "Взаимодействие Человека с Компьютером", август 1991

В этой статье мне хотелось бы остановиться на некоторых интересных с моей точки зрения идеях, связанных с возможностями современных компьютерных интерфейсов по трансформации знаний. Целью преобразования, которое должны осуществлять интерфейсы, является представление информации в виде, удобном для распознавания человеком. Например, человеку гораздо удобнее анализировать данные, классифицированные в виде дерева, чем работать с бесконечными столбцами цифр в таблице.

Предлагаемые вам, читатель, темы частично вызваны обсуждениями на Первом Московском Международном семинаре "Взаимодействие Человека с Компьютером" (Human-Computer Interface, HCI'91), прошедшем в Москве 5 - 9 августа 1991.

2. Почтенное собрание - мысли и участники

Нужно отметить, что международный компьютерный форум HCI'91, собравший около сотни участников из Европы, Азии и Америки выполнил свою главную задачу - организацию открытого обмена мнениями между Востоком и Западом. Было прослушано более 50 докладов и продемонстрировано около 25 систем разного рода.

Не удивительно, однако (учитывая, что это был первый семинар подобного рода в Москве), что исследователи с мировым именем представлены не были и извержения нетривиальных идей не получилось. Основной круг проблем, затронутых в докладах и дискуссиях, достаточно традиционен - интеллектуализация интерфейсов, объектно -ориентированный подход, визуализация, гипертексты и multi-media.

Центральная, на мой взгляд проблема - все возрастающее число различных пользовательских интерфейсов и необходимость их унификации путем реализации открытых систем с единой средой - при этом четко не прозвучала. Лишь несколько авторов посвятили этой проблеме достаточно внимания. Среди них, прежде всего - Steven Pemberton, руководитель пректа VIEWS. Хотелось бы отметить несколько наиболее интересных докладов, которые вызвали живой интерес аудитории и послужили началом плодотворных дискуссий.

В статье рассмотрены (с разной степенью подробности) следующие темы :

- открытые системы с единой средой управления и внесения изменений;
- свертка (folding) знания и информации при отображении на экран;
- выбор способа просмотра информации - скролинг против листания.

В качестве иллюстраций предложены конкретные системы :

- VIEWS, представленная Steven Pemberton, Holland (принцип TAXATA, единая среда редактирования).
- ASKER, представленная Андреем Свириденко, СССР (свертка информации и знания).

Book Window, представленная Ken-Ichi Okada, Japan (листание против традиционного скролинга - метафора книги).

Объем журнальной статьи позволяет мне лишь упомянуть среди других докладов исследование московских психологов И.В.Бурмистрова и А.Г.Шмелева, посвященное эффективности интерфейсов для систем с большим числом возможных планов действий и специфичными функциями. Одним из результатов этого исследования стали принципы реализации гибкого сценарного интерфейса (FSI - Flexible Scenario Interface), использующего сетевое представление сценария в виде графа, отображаемого на экране.

3. Открытые системы, единая среда

3.1. Проблемы

Чиçло различных пользовательских интерфейсов стремительно растет. Параллельная работа с разнообразными средами и интерфейсами часто приводит к ошибкам, путанице и неэффективной работе человека.

Для того чтобы бороться с проблемой разнообразия интерфейсов, существует несколько подходов, наибольшее распространение из которых получил подход "toolkit + guidelines".

Проблема состоит в том, что хотя toolkit (дословно - "пакет разработчика") и обязывает интерфейсы различных пользовательских программ быть похожими друг на друга, однако, такая реализация не может гарантировать того, что эти программы будут также единообразно использоваться.

Более эффективный подход состоит, видимо, в создании единой интерфейсной "прослойки" (layer) над всеми функциями и прикладными программами системы. Это следующий шаг по сравнению с уже ставшим традиционным разграничением прикладных и интерфейсных функций отдельной программы (рис. 1а).

Если выполнение команды обеспечивается некоторой "единой службой", то все программы могут эту команду использовать, не обладая о ней никакими дополнительными знаниями (рис. 1б). Например, оболочка (shell) системы Unix, выступая в качестве такой единой службы, позволяет всем программам использовать возможности команды "print *.doc" . В противоположность этому, в системе MS DOS, например, данная возможность является лишь частью "toolkit". Это позволяет использовать ее лишь тем программам, разработчики которых были достаточно предусмотрительны. Такой подход вынуждает пользователя помнить, какие команды доступны в данный момент, а какие - нет.

Другая проблема, возникающая в современных системах - недостаточная степень интеграции приложений, невозможность пользоваться всеми прикладными функциями системы из единой среды. Так, в системе Apple Macintosh, пионера в области multimedia, или в прекрасной издательской системе Ventura Publisher для IBM PC вы можете включать в документ текст и графические

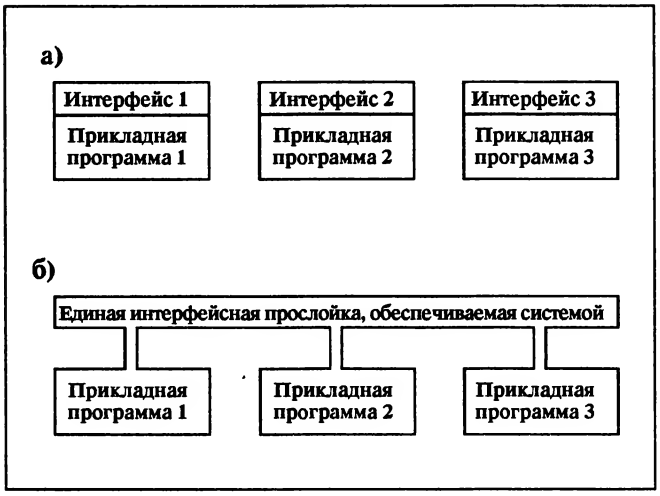


Рис. 1. Два подхода к проблеме соотношения интерфейса и прикладной программы

рисунки одновременно. Однако, для создания текста и создания рисунков приходится использовать различные программы. Собственно система предлагает альтернативу - текст с рисунками в нем или рисунки с текстом среди них. Оба варианта возможны и приводят к одинаковому результату. Однако, как только вы включили рисунки в свой текст, они оказываются "замороженными" - вы лишены возможности редактировать рисунок, кроме как полностью удалить его и заменить новым.

Кроме путаницы в работе пользователя, все эти проблемы приводят к невероятно высокой стоимости разработки интерактивных приложений. Решению проблемы пользовательского интерфейса может быть посвящено более 90 % кода программы.

Итак, были названы три проблемы:

- разнообразие пользовательских интерфейсов;
- обособленность и специфичность различных прикладных программ;
- высокая стоимость разработки пользовательских интерфейсов.

3.2. VIEWS - открытая система с единой средой

Один из подходов к решению названных проблем предпринят в проекте VIEWS - самом интересном, на мой взгляд, из представленных на конференции HCI'91. Основные идеи, лежащие в основе системы VIEWS, находящейся в стадии исследовательского прототипа, - это

- прямое манипулирование всеми объектами редактирования (direct manipulation), возможное на основе принципа TAXATA (Things Are eXactly As They Appear);
- реализация командного языка как функции редактора;
- открытая архитектура.

Принцип TAXATA является усилением известного принципа WYSIWYG (What You See Is What You Get).

В WYSIWYG-системах попытка отобразить на экране состояние объекта обычно реализована посредством представления копии редактируемого объекта, а не его самого. Таким образом, то, что вы видите (What You See) это то, что вам еще предстоит получить (What You WILL Get).

В TAXATA-системе VIEWS вы всегда работаете непосредственно с самим объектом, а не с его копией. Для борьбы со случайными нежелательными изменениями объекта или с "авариями" во VIEWS предусмотрены возможности неограниченной откатки (undo).

Основной принцип VIEWS можно сформулировать так : "ЛЮБОЙ объект в системе доступен для редактирования, ВСЕ действия реализуются посредством редактирования, и любой объект редактируется НЕПОСРЕДСТВЕННО". Примерами объектов могут служить текстовый документ, часы, диаграмма, меню и т.п. Так, список файлов в системе VIEWS является документом и вы можете осуществлять функции копирования, удаления и переименования файлов путем операций с соответствующими строками в редакторе. Точно также выполняются все другие виды заданий (tasks) без исключений.

Модель представления данных в системе VIEWS обеспечивает единую прослойку представления данных (data layer) для всех приложений, аналогично единому пользовательскому интерфейсу (user interface layer). Эта прослойка (data layer) позволяет прикладным программам осуществлять обмен объектами, без изменения их внутреннего строения (рис. 2).

3.3. Резюме

Главным достоинством системы VIEWS, которое хочется подчеркнуть еще раз, является тот факт, что как только вы научились основным приемам работы с редактором, вам становится очевидно, как обращаться с новым для вас приложением. Кроме того, нет необходимости дополнительно к редактору изучать командный язык.

4. СВЕРТКА - ОТ ПРЕОБРАЗОВАНИЯ К ПОНИМАНИЮ

Одна из интересных функциональных возможностей системы VIEWS - свертка информации. Впрочем, вопросы свертки более полно и глубоко разработаны в отечественной системе ASKER, представленной на семинаре докладом и демонстрацией.

Идеи VIEWS и ASKER вообще часто перекликаются, однако если во VIEWS основные усилия были направлены на создание открытой и единой среды для интеграции приложений, то ASKER планировался как посредник для работы с базой знаний. База знаний в данном контексте представляет собой сложный иерархический объ-

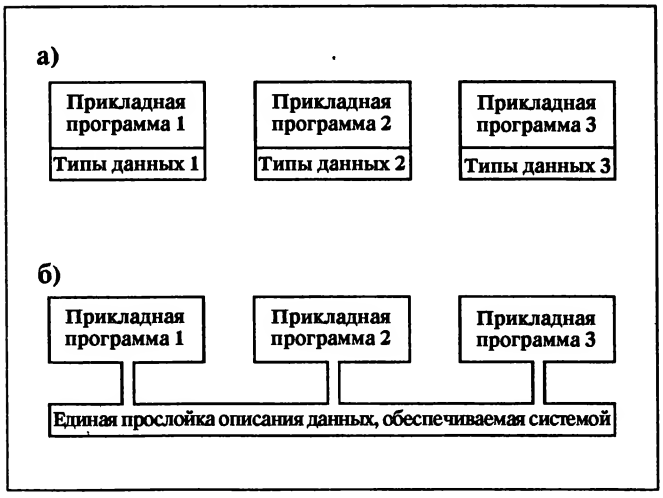


Рис. 2. Два подхода к проблеме представления данных

ект, имеющий чрезвычайно богатую систему внутренних связей.

4.1. Проблема

Любой достаточно сложный объект (например, длинный документ или реальная база знаний) не может быть осознан человеком одномоментно во всех его деталях. Люди умеют бороться с потоком обрушивающейся на них информации путем абстрагирования. Абстрагируясь, можно либо ограничить себя рассмотрением лишь небольшой части доступной информации, временно "забывая" о существовании других частей и связях между ними, либо отбросить все детали и получить весьма грубую, но целостную модель.

Кажется полезным реализовать интерфейсные функции, обеспечивающие как возможность "глобального взгляда" на создаваемый объект, так и отображение только деталей, существенных в данном контексте по запросу пользователя. Такие возможности будем называть сверткой. Понятия "свертка информации" и "фильтрация" являются синонимами в данном контексте. Рассматриваемое ниже понятие "свертка знания" несет в себе новое качество.

4.2. ASKER - от свертки информации к свертке знания

В системе VIEWS свертка (folding) информации представлена тем, что документ любой длины при обращении к системе должен помещаться в один экран. Достигается это тем, что автоматически демонстрируются лишь заголовки разделов документа. Таким образом, если представить документ как дерево, то на экран отображаются лишь верхние уровни его иерархии.

Более мощные возможности свертки реализованы в системе ASKER. Ручное управление сверткой информации в ASKER позволяет задавать гибкие смысловые фильтры. В системе предусмотрены интерактивные возможности для отображения или прятания (display / hide) любого уровня иерархии любого структурированного списка. Это позволяет пользователю ASKER создавать виды разной степени общности и расставлять необходимые в данном контексте акценты. Нечто подобное было реализовано в программе управления файловой системой Path Mainder, иногда использовавшейся до распространения Norton Commander.

Идею свертки можно суммировать в виде некого принципа отображения лишь необходимой информации - What You See Is What You Need (WYSIWYN) - "видно лишь то, что необходимо в данный момент". Принцип этот весьма прост и остается удивляться, почему никто не сформулировал его прежде. Впрочем, можно найти сходство свертки, как общего принципа, и таких идей как конструирование представлений в технологии СУБД, принцип Zoom, задание масок и фильтров, нелинейное преобразование информации в графических системах поддержки картографов и т.д.

Более интересна, по сравнению с простым удалением части информации с экрана (сверткой информации), является возможность свертки знания.

Свертка знания, в отличие от свертки информации, предполагает достаточно высокую интеллектуальность агента, который эту свертку производит. Так, например, данная статья может быть рассмотрена как свертка представленных на семинаре HCI'91 докладов и связанных с ними идей. В качестве простых примеров свертки знания могут быть названы вычисления по формулам, построение экстраполяций или интерполяций, получение следствий и выводов из известных посылок.

Суть свертки знания состоит в том, что определенная совокупность данных не исчезает с экрана совсем, а представляется в другом, значительно более емком и компактном виде. Свертка знания позволяет забыть о внутренней структуре объекта и манипулировать им как целостной единицей, обеспечивая легкость восприятия и экономичность мышления. Свертка знания представлена в ASKER возможностями формирования пороговых элементов (нейронов), имеющих несколько входов и несколько выходов.

Проиллюстрирую возможности свертки знания на весьма упрощенном примере. Пусть совокупность данных состоит из 6 единиц:

1. Процессор..... 80386
2. Тактовая частота..... 25 МГц
3. Оперативная память ..... 4 Мб
4. Винчестер..... 120 Мб
5. Монитор ..... Super VGA
6. Дисководы..... 3" и 5"

Тогда ее можно представить как новую сущность – "мощный персональный компьютер" с шестью входами и порогом 5. Эта новая сущность "сработает" (например, появится на экране) если в наличии любые 5 из 6 единиц данных.

Естественно, что одновременно со сверткой данных такого рода доджна быть предусмотрена и соответствующая "развертка" по запросу пользователя.

Продолжением свертки знаний в системе ASKER является процедура синтеза естественно-языковых текстов для отчета о результатах работы системы с данными. Текст является одним из наиболее привычных и удобных для человека способов свертки знания. ASKER трансформирует описание объекта, данное пользователем в терминах "признак - значение признака" в текст, представляющий собой набор возможных гипотез, относительно состояния данного объекта. Например, по симптомам больного ASKER синтезирует текстовое описание, представляющее собой результат классификации данного больного, и рекомендации по его лечению.

4.3. Резюме

Свертка знаний - это мощный механизм, который оказывается весьма эффективным для работы со сложными объектами, позволяя эффективно их структурировать. Свертка знания в системе ASKER, например, способна обеспечить поддержку информационного обмена между человеком и программной системой посредством понятий высокого уровня абстракции, понятий, имеющих (скрытую) сложную внутреннюю структуру. Нужно понимать, однако, что затраты на поддержку свертки знания в простых случаях (например для текстового редактора) могут себя не оправдать.

5. СВИТОК, КНИГА - ЧТО ДАЛЬШЕ ?

Со сверткой знания и информации связан также один из важных, с моей точки зрения, аргументов за скролинговый просмотр информации. Необходимость такой аргументации связана с тем, что существует другой удобный способ просмотра - пролистывание страниц (leafing through), как это делается в обычной книге.

До сих пор почти все программные системы для просмотра информации реализуют скролинговый подход. Этот подход находит свои корни в метафоре свитка, как хранилища информации.

Однако, развитие цивилизации постепенно привело к замене свитка книгой. Причина этого состоит, видимо, в том, что книга удобнее - позволяет осуществлять более эффективные способы доступа к хранящейся информации. Например книга, в отличие от свитка, позволяет пользоваться пространственными связями между объектами на одной странице.

Метафора книги реализована в системе Book Window, доклад о которой вызвал бурный отклик среди участников конференции. Система полностью эмулирует обычную книгу (вплоть до графической визуализации перелистывания страниц), предоставляя развитие метода доступа к информации (включая доступ по толщине) и поиска (включая поиск по фрагментам всех страниц).

Аргументация за пролистывание, обеспечивающее пространственные привязки, против скролинга, представляется мне важной, но не окончательной. Если принять во внимание возможности осуществления свертки информации и знания при скролинговом просмотре, то победитель не представляется очевидным. Мне лично потенциальные возможности скролинга кажутся более богатыми.

6. ОТ УДОБНОГО К ЕСТЕСТВЕННОМУ

Итак, возвращаясь к главной теме статьи - возможностям интерфейсов по трансформации знания, можно заключить, что эти возможности сегодня находятся в зачаточном состоянии, но, вероятно, будут весьма перспективны завтра. Уже сегодня стоит вопрос о реализации одного из способов трансформации знания - свертке.

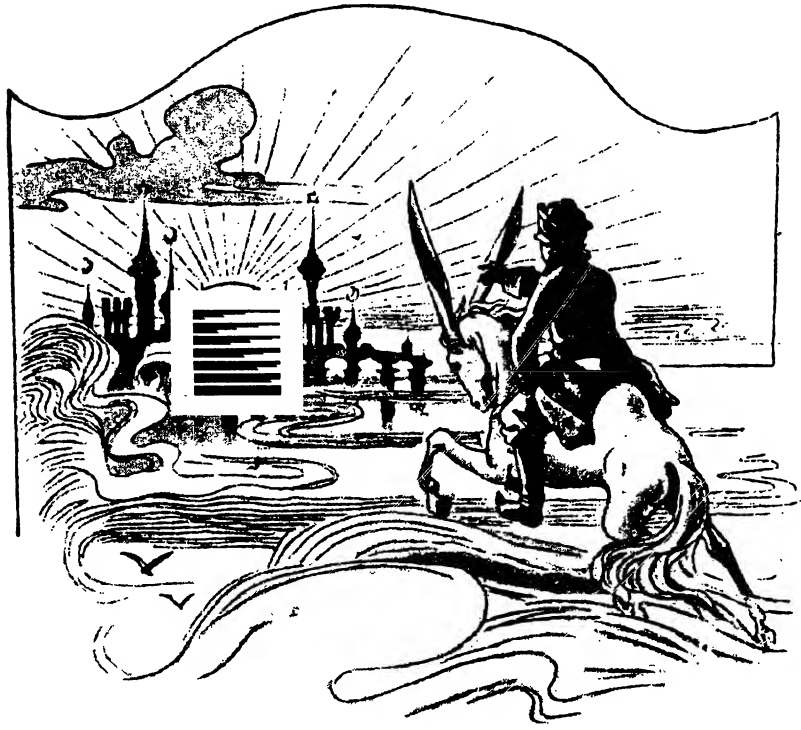
Очевидно, что сегодняшние ограничения интеллектуальности интерфейсов связаны в первую очередь с громадной сложностью интеллекта. Нужно честно признаться, что сегодня мы недостаточно понимаем как устроен и как функционирует самый сложный объект во вселенной - человеческий мозг. Недостаток интеллектуальности программных систем восполняется сегодня приписыванием этого модного слова многим не относящимся к делу возможностям.

Сегодня, однако, четко стоит вопрос о реализации удобного и естественного пользовательского интерфейса.

С моей точки зрения, программная система обладает удобным пользовательским интерфейсом, если пользователь располагает средствами определить цель взаимодействия быстро ("быстро" и "легко" здесь синонимы) и может быстро получить необходимый ответ в удобной для себя форме. Удобный интерфейс с одной стороны максимизирует вероятность и минимизирует усилия, необходимые для достижения полезного состояния, а с другой - минимизирует вероятность попадания в опасную или бесполезную ситуацию.

Естественный интерфейс должен быть с одной стороны удобным, а с другой - привычным, то есть не разрушать стереотипы пользователя и не нарушать его психологический комфорт.

Nantucket делает ставку на советский рынок



Clipper Summer '87 не нуждается в представлении. Это, пожалуй, самая популярная инструментальная система для разработки прикладных баз данных. Большинство программ типа "Бухгалтерия", "Кадры" и т.д., созданных в СССР, написаны именно при помощи компилятора Clipper. Его преемник, Clipper 5.0, появился совсем недавно, но уже сейчас можно сказать, что новая версия повторит историю успеха Clipper Summer '87.

В то время как технические достоинства системы Clipper известны хорошо, стратегия фирмы Nantucket, ее маркетинговая политика неизвестны большинству читателей. Недавно редактор "Монитора" Андрей Орехов взял интервью у Вирджинии Клаф, Директора по Оперативному Управлению в СССР. Они беседовали не о системе Clipper, а о том как фирма Nantucket делает свой бизнес в СССР.

Орехов: *Первый вопрос. Зачем Nantucket пришла в Россию? Неужели здесь работать проще, чем, скажем, в Европе?*

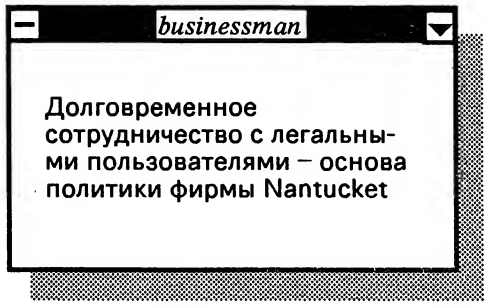
Клаф: Конечно, условия, в которых работает Nantucket Corp., разные в США и в России. Но дело не в этом. Nantucket всецело нацелена на международные рынки. Мы обладаем семью филиалами в разных странах мира, доля международного бизнеса в нашей деятельности составляет примерно половину. Поэтому как только появилась возможность, Nantucket сразу же начала работать в Советском Союзе. К этому времени большое число программистов в Советском Союзе уже использовали Clipper, но, конечно, нелегальные копии. Мы хотели дать им возможность купить продукт и обеспечить технической поддержкой, которую имеют пользователи Clipper во всем мире. Кроме того, Советский рынок - это последний нео-

своенный рынок в мире, это понимают все серьезные бизнесмены.

О: *Какими были первые шаги Nantucket в СССР?*

К: Мысль о ведении бизнеса в Советском Союзе впервые зародилась в Nantucket Corporation осенью 1989 года. Ванесса Вейд (Управляющий директор представительства Nantucket в Брюсселе) и Бэрри Рибелл (владелец и глава Nantucket) приехали в Москву с целью выяснения обстановки. Nantucket имеет достаточный опыт ведения дел на международном рынке, однако то, что мы здесь увидели, поразило нас. Примерно 40-60 тысяч нелегально используемых копий Clipper. Талантливые программисты, создающие программы без документации, технической поддержки и "ноу-хау" маркетинговой деятельности. Отсутствие дистрибьюторской сети для распространения готовых продуктов на рынке. Это был подходящий момент для "прыжка" Nantucket на советский рынок.

Проведя несколько месяцев в поисках потенциальных партнеров, Nantucket сделала следующее. В начале 1990 года, она стала первой западной фирмой по производству программного обеспечения, которая начала продажи и открыла центр поддержки в Москве при содействии компьютерного центра Кировского района "Кировец". В ноябре 1990 года отношения между двумя организациями были официально оформлены путем создания совместного предприятия "Магнит". И вновь Nantucket оказалась первой компанией по производству программного обеспечения, которая предприняла такой шаг и направила в Советский Союз американцев для совместной работы с советскими специалистами с целью оказания помощи в развитии бизнеса.



О: *Nantucket одной из первых начала продажу своих продуктов за рубли. Но для западной фирмы советские деньги не представляют серьезной ценности. Как Вы собираетесь зарабатывать валюту?*

К: Пока мы не рассчитываем на валютную прибыль, хотя некоторое количество Clipper продано за валюту. Сейчас всю получаемую прибыль мы реинвестируем в развитие нашего бизнеса в СССР. Эти деньги идут на финансирование разработок программного обеспечения, обучение и поддержку зарегистрированных пользователей, на открытие Clipper-клубов, рекламу и многое другое. Nantucket собирается работать в Советском Союзе не один год. Поэтому мы вкладываем деньги сейчас, чтобы остаться лидирующей компанией в области баз данных и через пять лет.

О: *В чем заключается стратегия деятельности Nantucket в Советском Союзе?*

К: Стратегия определилась сразу же. Поскольку целью деятельности СП "Магнит" является содействие развитию советского рынка программного обеспечения и в связи с этим увеличению числа зарегистрированных пользователей Clipper, необходимо предлагать полный набор продуктов Nantucket и услуг по поддержке на русском языке и за рубли.

Первым делом СП "Магнит" занялось русификацией Clipper-продуктов. Пользователи Summer'87 и Tools One получили всю документацию на русском языке. Следующим шагом является недавно подготовленная русифицированная версия Clipper 5.0. Здесь, в дополнение к полной документации на русском языке, были также переведены все утилиты, экранные команды, функции, встроенная документация Norton Guide.

О: *Несколько слов об организации продажи программного обеспечения. К сожалению, у нас этому вопросу уделяется неоправданно мало внимания. Хотя очевидно, что результат любой деятельности, будь то производство программ или автомобилей, проявляется только при продаже этого товара на рынке.*

К: Совершенно верно. Продажа программного обеспечения, даже такого как Clipper - дело чрезвычайно непростое. Высокое качество продукта лишь необходимое условие для успешной продажи, но не достаточное. Для организации сбыта требуются деньги, реклама, налаженная дилерская сеть, талантливые люди и, наконец, правильная маркетинговая политика.

О: *Какая политика у фирмы Nantucket?*

К: Главное правило - долговременное сотрудничество с зарегистрированными пользователями. После совершения продажи наши отношения с покупателем не заканчиваются, они только начинаются.

Чтобы действительно внести изменения в природу здешнего рынка и гарантировать пользователям возможность применять все преимущества наших продуктов, СП "Магнит" должно поддерживать своих клиентов.

Предоставляются услуги различных видов. Фирма обеспечивает своих верных пользователей доработанными версиями (update) и модификациями, а также предоставляет значительные скидки при приобретении последующих Clipper-продуктов.

Любой зарегистрированный пользователь может получить поддержку по применению Clipper по, так называемой, "горячей линии" по телефону. Для решения более сложных вопросов легальные пользователи могут лично проконсультироваться со специалистами Центра поддержки в Москве.

В семи городах Советского Союза действует 9 сертифицированных центров обучения, где программисты могут научиться наиболее эффективным приемам программирования. Обучение ведут преподаватели, прошедшие подготовку на фирме.

СП "Магнит" организовало в Москве, Ленинграде и Минске группы пользователей, получившие название Clipper-клубы. В ближайшие месяцы планируется создать еще несколько таких групп. Участвуя в заседаниях Clipper- клубов, организуемых два раза в месяц, наши пользователи имеют возможность услышать свежую информацию от Nantucket, обменяться друг с другом собственными идеями и получить квалифицированную

оценку своих последних разработок от опытных программистов, работающих на Clipper.

Кроме того, каждый зарегистрированный пользователь Clipper бесплатно получает подписку на журнал Nantucket News, издаваемый на русском языке, в котором можно прочитать информацию о применении Clipper.

О: *Госпожа Клаф, каким образом Nantucket решает проблемы продажи программного обеспечения на такой огромной территории, как СССР?*

К: Когда мы начали свой бизнес в СССР, оказалось, что здесь отсутствуют хорошие каналы распространения. СП "Магнит" должно было создать свою собственную дилерскую сеть. Требовалось найти надежных дилеров, организовать их обучение, наладить обратную связь и поставку продукции. В настоящее время "Магнит" имеет в самых разных городах Советского Союза 30 хорошо зарекомендовавших себя дилеров, которые занимаются активными поисками покупателей Clipper в своих регионах и обеспечивают их настоящими фирменными копиями наших продуктов. От этих дилеров мы получаем информацию о состоянии рынка и пожеланиях советских программистов, учитываемых при разработке будущих версий Clipper.

О: *Нужны ли Вам еще дилеры?*

К: Да, конечно. Мы заинтересованы в расширении своей дилерской сети. Чем больше будет дилеров, тем больше программистов смогут купить Clipper, а затем получить настоящую фирменную поддержку, не выезжая из своего города или района.

О: *Какие требования предъявляются к будущему дилеру? Неужели им может стать каждый?*

К: Фирма Nantucket открыта для сотрудничества. Дилером может стать любая фирма или частное лицо. Главное, чтобы у будущего дилера было горячее желание продавать Clipper, обеспечивать поддержку зарегистрированным пользователям в своем регионе.Дилер получает необходимую подготовку в наших центрах обучения, а в ряде случаев может рассчитывать на финансирование работ по сбыту и разработке прикладных систем.

О: *Несколько слов о "пиратстве". В Союзе ворованных копий Clipper намного больше, чем легальных. Как Вы боретесь с подобными явлениями?*

К: Конечно, мы прикладываем много сил для борьбы с "пиратством". Недавно несколько ведущих советских и западных фирм, в том числе Nuntucket, объявили об организации в Москве Ассоциации поставщиков программных продуктов, одной из задач которой является борьба с "пиратством". Трудности на пути решения этой проблемы могли бы стать темой для отдельного интервью. Поэтому я ограничусь тем, что сказала, а вообще мы значительно большую часть энергии расходуем на поддержку зарегестрированных пользователей.

Все услуги, о которых я говорила, предоставляются только легальным пользователям Clipper-продуктов. Nantucket очень строго придерживается этой политики, поскольку зарегистрированные пользователи заплатили за легальные копии нашего продукта и тем самым внесли плату и за услуги.

Кроме того, программисты, пытающиеся написать программу, используя нелегальную копию, обречены на неудачу. Во-первых, им закрыт путь на западные рынки, где за соблюдением авторских прав контролируется очень жестко. Вероятно, через некоторое время такое положение будет и в Советском Союзе. Во-вторых, без использования преимуществ предоставляемой поддержки и уже имеющихся методов и средств, они не смогут создать первоклассный продукт и поэтому попусту тратят время, силы и деньги.

О: *И, наконец, последний вопрос. Кто является Вашим конкурентом на советском рынке?*

К: Я бы сказала, что рынок в Советском Союзе настолько огромен, что у Nantucket фактически здесь нет конкурентов, за исключением, пожалуй, "пиратов". Мы предпочитаем заниматься здесь созидательной работой. Хотим содействовать советским программистам в создании собственного рынка. Помочь им работать в манере цивилизованного бизнеса. Под этим мы подразумеваем сочетание достижений западной маркетинговой деятельности и "ноу-хау" с российским искусством программирования, чувством здравого смысла в оценке местных условий и желанием добиться успеха. Мы полагаем, что создание инфраструктуры для советского рынка программного обеспечения значит больше, чем просто деятельность Nantucket и распространение системы Clipper.

О: *Благодарю Вас за столь содержательную беседу о политике фирмы Nantucket с читателями "Монитора".*

К: Пожалуйста.

Адрес СП "Магнит": СССР, 127018, Москва, 2-я Ямская, 15.Телефон: (095) 289-44-77 или (095) 289-44-83 Факс: (095) 289-34-69

Список продуктов Nantucket Corp. (состояние на 1 ноября 1991 года)

Clipper 5.0 (англоязычная версия)
Clipper Summer'87, Clipper Tools One (русскоязычная версия, с правом 25 % скидки при покупке Clipper 5.0)
Clipper Tools One (русскоязычная версия, только для официальных пользователей Clipper Summer'87)
Clipper 5.0 (русифицированная версия)

Количество проданных копий Clipper (состояние на 1 ноября 1991 года)

В мире	**более 200.000 копий**
В том числе в СССР	**2.500 копий**

Из книги рекордов Гиннеса

Самым первым программируемым компьютером был 1500-ламповый "Колосс", спроектированный профессором М.Ньюменом (1897-1985) и построенный Т.Х.Флауэрсом. Начал действовать в декабре 1943 г., в Блетчли Парк, Бакингемшир, для расшифровки германской шифровальной машины "Энигма". Он появился в результате концепции, опубликованной доктором А.Тьюрингом (1912-1954) в его работе "Исчисляемые числа с применением в проблеме решений." "Колосс" был рассекречен 25 октября 1975 года.

Первым компьютером с накопителем программ был "Марк-1" Манчестерского университета, в котором применена катодная лучевая трубка Уильямса (патент 11 декабря 1946 г.). Первая программа была выполнена на нем профессором Томом Килбургом (род.1921) за 52 мин.

Изобретение точечно-контактного транзистора Д.Бардином и У.Брэттейном, о чем было объявлено в июле 1948 г., и плоскостного транзистора Р.-Л. Уоллесом, М.Спарксом и доктором У.Шокли (род.1910) в начале 1951 г. дало мощный импульс развитию компьютеров. Концепция интегральной схемы, приведшая к развитию микроминиатюризации, была впервые опубликована 7 мая 1952 г. Д.Даммером (род.1909). Микрокомпьютер был получен в 1969-1973 гг. в связи с изготовлением микропроцессора "4004" М.-Э.Гоффом-младшим из корпорации "Интел Корпорейшн".

18 июля 1980 г. Шакунтала Деви из Индии продемонстрировала умножение двух 13-значных чисел 7 686 369 774 840 * 2 465 099 745 779, выбранных наугад отделом ЭВМ Имперского колледжа в Лондоне. Операция заняла у нее 28 секунд. Ее правильный ответ был равен 18 947 668 177 995 426 462 773 730.

Одним из самых мощных и быстродействующих компьютеров является "CRAY-2" с жидким охлаждением, названный так по имени С.Крея из компании "Крэй Рисерч Инкорпорейтед", Миннеаполис, Миннесота. Емкость памяти 256 млн. 64-битовых слов или 32 млн. байтов в основном блоке памяти. Быстродействие - 250 млн. операций в секунду. Стоимость системы со средним объемом памяти - 17 млн. долл. Один из наиболее мощных компьютеров Великобритании - Интернэшнл Компьютерс Дистрибьюшн Эррей Процессор - ICLDAP.

Мегабитный барьер был преодолен четырьмя японскими компаниями "Хитачи", "NEC", "NTT Ацуги Электрикэл Коммюникейшнз" и "Тошиба" в феврале 1984 г., когда ими была изготовлена интегральная схема "1024-бит" размером с головку стержня шариковой ручки и толщиной с человеческий волос.

Самой крупной фирмой по производству ЭВМ является "Интернэшнл Бизнесс Машинз Корпорейшн" (IBM), находящаяся в Нью-Йорке. В декабре 1986 года активы компании оценивались в 57 814 000 долл. и валовый доход в 51 250 000 долл. В октябре 1979 г. компанией был сделан самый крупный в истории этой корпорации заем в размере 1 млрд. долл. На компанию во всем мире трудятся 403 508 служащих, количество акционеров составляет 792 689.

Человеческий компьютер

Быстрее других, за 1 мин. 28 с., извлек корень 13 степени из числа из 100 цифр датчанин Виллем Клейн (1912-1986) 7 апреля 1981 г., в национальной лаборатории в Цукуба, Япония.

"IBM не рассматривает рекламу как лишнюю статью расходов, а расценивает ее как выгодное помещение капитала..."

Фрэнсис Дж. Роджерс,
вице–президент корпорации IBM по маркетингу

Если Вам не все равно, кто читает Вашу рекламу, то обратите внимание на журнал "Монитор"

"Монитор" читают люди, которые являются потенциальными покупателями Ваших программ и апппаратуры

О Вашей фирме узнают сотни предприятий страны, которые стали подписчиками "Монитора"

Над Вашим объявлением будут работать профессионалы

Реклама в "Мониторе" – выгодное помещение капитала

Факс (095) 126–31–56

Адрес: 117419, Москва, а/я 765.

IBM зарегистрированная торговая марка International Business Machines Corporation

ПП "Офсет" МГПО. Зак. 1556

Софт·Москва

СПРАВЕДЛИВЫЕ ЦЕНЫ – ОСНОВА НАШЕГО БИЗНЕСА

Брокерская группа "Софт–Москва"
предлагает все необходимое для работы

- компьютеры XT/AT–286/386/486;
- принтеры, плоттеры, сканнеры, дигитайзеры, стриммеры;
- ксероксы, телефаксы, модемы;
- телефонные аппараты.

Справедливые цены–основа нашего бизнеса. Обширная информационная база позволит предложить Вам цены, соответствующие реальному качеству товара.

Для оптовых покупателей предоставляется скидка.

Для розничных покупателей полный комплекс работ по тестированию и пуско–наладке.

Для всех – бесплатные консультации.

ТЕЛЕФОН/ФАКС (095) 126–31–56
ДЛЯ ПИСЕМ: 117419, Москва, а/я 765.